Developpez Le Mag

Édition d'août – septembre 2014
Numéro 53
Magazine en ligne gratuit
Diffusion de copies conformes à l'original autorisée
Réalisation : Alexandre Pottiez & Sébastien Lataix
Rédaction : la rédaction de Developpez
Contact : magazine@redaction-developpez.com

Sommaire

Éditorial

Le magazine des développeurs est de retour avec de nouveaux articles, de nouvelles rubriques, de nouvelles news. La rédaction est heureuse de vous présenter un florilège de ses meilleures ressources.

La rédaction

Article 2D/3D/Jeux

Commençons... avec des cercles

La création de ses propres graphismes est une nécessité pour la plupart des développeurs indépendants. Nous allons essayer de commencer avec des idées basiques et des exercices pour améliorer ces notions.
par Alexandre Laurent
Page 70

Article Réseau

Modélisation en couche des protocoles réseau

L'objectif de cet article est d'introduire les concepts de modélisation, de présenter les deux modélisations dominantes et le « dénominateur commun » qui en est issu.
par Philippe Latu
Page 93

Assembleur

Les derniers tutoriels et articles

L'assembleur en ligne

Ce tutoriel va vous présenter l'assembleur en ligne avec le langage C et le compilateur

1 Introduction

Aujourd'hui, le langage assembleur est assez peu utilisé. La plupart des programmeurs utilisent les langages haut niveau, comme le C ou le C++, pour plusieurs raisons. En premier lieu, parce que ces langages permettent d'écrire des programmes indépendants de l'architecture, et donc portables. En second lieu, parce qu'ils présentent des syntaxes simples et compréhensibles, ce qui permet d'augmenter la productivité, lors de l'écriture et la maintenance du code. Mais, de temps en temps, les programmeurs ont besoin d'utiliser des instructions assembleur dans leurs programmes.

L'assembleur en ligne, *Inline Assembly*, est une extension des langages de programmation haut niveau standard offerte par certains compilateurs. Il permet d'inclure des instructions assembleur dans un programme écrit en langage haut niveau.

Ce tutoriel va vous présenter l'assembleur en ligne avec le langage C et le compilateur *GCC*. Les différents exemples cités sont téléchargeables sur Developpez.com : lien 1.

2 Quand utiliser l'assembleur en ligne ?

L'ajout des instructions assembleur dépendantes de l'architecture à un programme écrit dans un langage haut niveau affecte sa portabilité. Ainsi, vous ne devriez utiliser des instructions assembleur qu'en dernier ressort. Par exemple, lorsque vous constatez que l'utilisation de l'assembleur va optimiser vos codes et vous produira un programme rapide, ou lorsque vous voulez utiliser des instructions spécifiques à l'architecture et non descriptibles par une syntaxe de haut niveau.

En fait, la capacité d'optimisation des compilateurs est limitée à cause de plusieurs facteurs, tels que leur compréhension limitée du comportement du code, et le contexte dans lequel il sera utilisé, et aussi le besoin des programmeurs d'effectuer la compilation rapidement. En effet, la tâche d'optimisation est coûteuse en mémoire et en temps de compilation. Ainsi, la substitution des parties complexes du code et sensibles à la performance du programme par des simples instructions assembleur va simplifier l'optimisation d'une part, et permettra de produire un programme performant et rapide, d'autre part.

En outre, plusieurs fonctionnalités des processeurs sont encore difficiles à décrire par une syntaxe de haut niveau. Ainsi, plusieurs instructions machine doivent être écrites à la main. Parmi celles-ci, on cite les instructions système d'accès aux ports d'entrée/-sortie (*x86* : instructions *OUT*, *OUTS*, *IN*, *INS*). On peut aussi citer les instructions de gestion de la mémoire (*x86* : *LLDT*, *LGDT*...) et des tâches (*x86* : *LTR*).

L'utilisation des instructions assembleur nécessite une connaissance avancée de l'architecture et du fonctionnement interne du compilateur. En effet, l'assembleur en ligne est largement utilisé par les développeurs des systèmes d'exploitation et des pilotes de périphériques (*drivers*).

Dans ces domaines de programmation, la portabilité d'un code n'est pas une exigence extrême et n'a parfois pas de sens. En effet, le code source d'un driver ou d'un système d'exploitation ne pourra jamais être portable. La haute priorité est, souvent, donnée à la production d'un programme performant qui s'exécute rapidement et qui permet d'exploiter efficacement toutes les fonctionnalités d'une architecture donnée.

Pour les systèmes d'exploitation, le terme **multiplate-forme** est plus précis. Par exemple, le système Linux est multiplate-forme parce qu'il peut tourner sur une variété d'architectures. En fait, dans le code source de son noyau, chaque architecture possède sa propre description en langage C et en assembleur (*/usr/src/linux/arch/*). Cela vient du fait que les instructions et les caractéristiques diffèrent

d'une architecture à une autre. Autrement dit, le code source dépendant d'une architecture ne peut pas être compilé pour tourner sur une autre.

Les instructions assembleur sont souvent encapsulées dans des macros ou des fonctions en ligne définies dans des fichiers d'en-tête. C'est pour faciliter la maintenance du code source et pour faciliter son portage d'une architecture à une autre. Consultez le répertoire */usr/src/linux/arch/x86/include/asm* pour avoir accès à des exemples de code contenant des instructions assembleur *x86*. Par exemple, le fichier *io.h* contient les instructions *x86* d'accès aux ports d'entrée/sortie. En outre, le fichier *string.h* contient l'implémentation des opérations de manipulation des chaînes en utilisant le jeu d'instructions *x86*.

Dans ce tutoriel, on va étudier quelques exemples des codes assembleur pris du code source du noyau *Linux 0.01*. Vous pouvez le télécharger ici : lien 2.

3 L'assembleur en ligne avec le langage C

En particulier, le compilateur *GCC, GNU Compilers Collection* dispose d'une syntaxe simple et complète permettant d'utiliser efficacement des instructions assembleur dans un programme C.

Dans un programme assembleur classique, chaque instruction possède, typiquement, deux opérandes : un opérande source et un opérande destination. Ces opérandes peuvent être utilisés de façon explicite ou implicite. On peut citer comme exemple l'instruction *x86 MOV*, qui utilise explicitement deux opérandes. En utilisant la syntaxe *AT&T*, on peut écrire cette instruction comme suit :

```
1  MOV source, destination
```

Les opérandes peuvent être de trois types : registre, mémoire ou immédiat. Dans l'exemple ci-dessous, l'instruction *MOV* va transférer le contenu du registre *EDX* dans *EAX* :

```
1  MOV EDX, EAX
```

Avec l'assembleur en ligne, on ne va pas réinventer la roue, croyez-moi ! Les instructions seront écrites de la même façon, sauf que les opérandes seront définis séparément et leurs valeurs pourront être des expressions C. Autrement dit, tous ce que vous avez appris sur l'assembleur restera valide et vous en aurez besoin. **Vous allez juste apprendre une syntaxe qui vous permettra d'écrire des instructions assembleur dans un programme C.**

Avec le compilateur *GCC*, les instructions assembleur, ainsi que la définition de leurs opérandes, doivent être encapsulées dans une construction déclarée, typiquement, avec le mot clé asm. Cela va indiquer au compilateur que le code à l'intérieur doit être traité d'une façon différente. Pendant la compilation, le compilateur *GCC* va utiliser les informations incluses dans la construction *asm* pour générer les instructions assembleur et les placer dans le code-cible.

Notez bien que GCC utilise par défaut la syntaxe *AT&T* pour générer le code-cible. En effet, le compilateur utilise, par défaut, le programme *as* (*GNU Assembler*) pour générer le code objet. Ainsi, dans la construction *asm*, les instructions assembleur doivent être écrites en utilisant cette syntaxe, à moins que vous n'utilisiez l'option *-masm* du compilateur. Dans ce qui suit, on va utiliser la syntaxe AT&T. Voici un tutoriel qui va vous aider à l'apprendre rapidement : lien 3

4 Syntaxe de la construction asm

Le mot clé *asm* doit être suivi par une expression entre parenthèses et constituée de sections séparées par deux points. La première section contient des instructions assembleur écrites entre guillemets. Dans la deuxième, on doit spécifier les opérandes de sortie des instructions. La troisième section contient les opérandes d'entrée. La quatrième section sert à déclarer les modifications apportées, sur les registres ou en mémoire, par les instructions.

Important : vous devez toujours placer les deux points qui séparent la deuxième section de la troisième, même si la section de sorties est vide.

Le programme suivant utilise l'instruction assembleur *movl* pour affecter la valeur de la variable *b* à variable *a* (*a = b ;*) :

```
1  #include <stdio.h>
2  int main(void)
3  {
4   int a = 10;
5   int b = 5;
6
7   __asm__("movl\t%1, %0"
8
9           : "=&r" (a) : "r" (b)
10          : /* liste des modifications */
11         );
12
13  printf("a = b = %d \n", a);
14  return 0;
15 }
```

Ici, chaque opérande est écrit comme une expression C placée entre parenthèses et précédée d'une chaîne de caractères indiquant sa contrainte. La lettre *r*

dans chaque contrainte symbolise le nom d'un registre général du processeur. Ainsi, le compilateur *GCC* va allouer un registre de ce type, pour stocker la valeur (5) de la variable *b*, et un autre pour stocker celle (10) de *a*. Le caractère « = » dans la contrainte *"=r"* du premier opérande indique que c'est un opérande de sortie. Toutes les contraintes des opérandes de sortie doivent commencer par « = ».

Note : le caractère « & » dans la première contrainte va indiquer au compilateur de ne pas allouer le même registre pour les deux opérandes. L'utilisation de ce caractère dans l'exemple ci-dessus n'est pas obligatoire, mais dans certains cas il est très important. On va expliquer, en détails, son utilisation dans la section 4.5. Déclaration des modifications.

Pour se référer aux opérandes dans le code assembleur, on peut écrire leurs indices précédés du caractère « % ». Ainsi, l'instruction *movl* va transférer le contenu (5) du deuxième opérande (source ou entrée d'indice 1) dans le registre alloué au premier opérande (destination ou sortie d'indice 0).

Important : la production de sorties aura lieu, toujours, après l'exécution de la dernière instruction du code assembleur.

Dans le code ci-dessus, après l'exécution de l'instruction *movl*, la sortie *a* sera produite. C'est en affectant le contenu du registre destination à la variable *a*.

GCC ne peut pas analyser les instructions assembleur et vérifier si elles sont valides ou non. Ainsi, il ne peut pas vérifier si le type des opérandes est raisonnable pour les instructions ou non. En fait, c'est le rôle de l'assembleur *as*. En ce qui concerne la construction *asm*, *GCC* va juste, dans un premier temps, vérifier la syntaxe des expressions C utilisées comme opérandes. Ensuite, il va générer les opérandes des instructions en utilisant les informations contenues dans les sections d'entrées et de sorties. Finalement, il va placer les instructions assembleur avec leurs opérandes dans le code-cible.

Enfin, pour obtenir le code-cible produit par le compilateur *GCC*, faites-lui passer l'option *-S*. Faites passer, également, l'option *-O2*, pour appliquer le deuxième niveau d'optimisation de *GCC* à votre code. Voici une portion du code-cible de l'exemple ci-dessus :

```
1         movl    $5, %edx
2         movl    %esp, %ebp
3         .cfi_def_cfa_register 5
4         andl    $-16, %esp
5         subl    $16, %esp
6 #APP
7 # 9 "movl.c" 1
8         movl    %edx, %eax
9 # 0 "" 2
10#NO_APP
11        movl    $.LC0, (%esp)
12        movl    %eax, 4(%esp)
13        call    printf
14        xorl    %eax, %eax
15        leave
```

Dans le code-cible, le bloc *asm* est placé entre les deux lignes de commentaires *#APP* et *#NO_APP*. Le compilateur a alloué deux registres généraux pour stocker les valeurs de *a* et *b* : le registre *EAX* pour *a* et le registre *EDX* pour *b*.

4.1 Les instructions assembleur

La première section de la construction *asm* contient les instructions assembleur. Dans cette section, vous devez spécifier un modèle d'instructions assembleur, un peu comme ce qui apparaît dans une description machine (un fichier *.s*).

En utilisant la syntaxe *AT&T*, si vous voulez utiliser le nom d'un registre comme opérande, préfixez-le par « %% ». Pour se référer à un opérande d'entrée ou de sortie, dans une instruction assembleur, écrivez le caractère « % » suivi de son indice (0, 1... 29).

Comme dans un code assembleur classique, les instructions de *asm* doivent être séparées par des séparateurs. En assembleur GNU, on a le choix d'utiliser soit le caractère « *;* », soit le caractère saut de ligne. Si vous choisissez ce dernier, placez « $\backslash n$ » après chaque instruction, sauf la dernière.

Notez que vous pouvez mettre plusieurs instructions dans la même chaîne, en les séparant par des « *;* », comme suit :

```
1 __asm__("\tnop;nop;nop\t"
2         "jmp \t1f\n"
3         "1:" ::);
```

Le « $\backslash n$ » sera traduit en un saut de ligne dans le code-cible. Le caractère « $\backslash t$ » est optionnel et il sera traduit en une tabulation.

Pendant la compilation, *GCC* utilise les informations (sur les opérandes) contenues dans les sections d'entrées et de sorties pour générer les opérandes des instructions assembleur.

4.2 Les opérandes

Chaque opérande peut être écrit comme une expression C placée entre parenthèses et précédée d'une chaîne de caractères indiquant sa contrainte. En général, les instructions assembleur utilisent trois types d'opérandes : les opérandes registre, les opérandes mémoire et les opérandes immédiats. Dans la construction *asm*, les contraintes vont préciser si un opérande doit être stocké dans un registre, et

quel type de registre ; si l'opérande doit être une référence dans la mémoire, et quel type d'adresse utilisé ; si l'opérande est une constante immédiate, et quelle valeur elle peut avoir. Une contrainte peut être constituée d'un ou plusieurs modificateurs et lettres.

4.2.a Les lettres

Les lettres spécifient souvent les registres d'une architecture donnée. Le tableau suivant liste les lettres couramment utilisées avec l'architecture *x86* :

Lettre	Registre
R	*EAX*, *EBX*, *ECX*, *EDX*, *ESI*, *EDI*, *ESP* et *EBP*
q	*EAX*, *EBX*, *ECX*, *EDX* (mode 32 bits)
a	Registre *EAX*
b	Registre *EBX*
c	Registre *ECX*
d	Registre *EDX*
S	Registre *ESI*
D	Registre *EDI*
A	Combinaison *EAX :EDX*

Il existe aussi un ensemble de lettres commun à toutes les architectures. Consultez le manuel de *GCC* (à cette adresse : lien 4) pour avoir accès à la liste complète de contraintes.

Avec l'architecture *x86*, on utilise souvent la contrainte « *r* » ou « *R* » pour spécifier un opérande registre, la contrainte « *m* » pour spécifier un opérande mémoire et la contrainte « *i* » ou « *I* » pour spécifier un opérande immédiat.

En outre, un nombre (0, 1, 2... 9) peut être utilisé dans la contrainte d'un opérande d'entrée, pour dire que cet opérande fait référence à un opérande de sortie. Ainsi, le compilateur va allouer la même *location* pour stocker les valeurs des deux opérandes.

4.2.b Les modificateurs

Pour les modificateurs, on a déjà vu le modificateur « = ». Il en existe cinq autres, parmi lesquels on cite les modificateurs « + » et « & ». Le premier indique que l'opérande est en écriture et en lecture à la fois. Le modificateur « & » a une utilisation spécifique, qu'on va expliquer dans les sections suivantes.

4.3 Les sorties

La deuxième section de la construction *asm* contient les opérandes de sortie, séparés par des virgules. Chaque opérande doit être une expression C de type *lvalue*. C'est-à-dire qu'on peut le mettre à gauche d'une affectation. Par exemple, vous ne devez pas utiliser une constante déclarée avec la directive *#define* ou une variable C de type *const* comme opérande de sortie. Le compilateur vérifie cela pour chaque opérande de sortie.

Un opérande de sortie ordinaire est en écriture seulement. Ainsi, les contraintes de sortie contiennent, souvent, le modificateur « = ». La valeur d'un tel opérande reste indéterminée jusqu'à la production des sorties ! Vous pouvez vérifier cela dans l'exemple *movl.c*, en inversant l'ordre des opérandes de l'instruction comme suit :

```
1  __asm__("movl\t%0, %1"
2
3          : "=&r" (a)
4          : "r" (b)
5          :/* liste des modifications */
6         )
```

Ainsi, si vous décidez d'utiliser un opérande de sortie en lecture et en écriture, vous devez utiliser le modificateur « + » à la place de « = ».

4.4 Les entrées

Les opérandes d'entrée occupent la troisième section de la construction *asm*. Ils sont traités de manière différente de celles de sortie, bien qu'ils aient la même syntaxe.

Contrairement aux opérandes de sortie, un opérande d'entrée est par défaut en lecture et en écriture (sans utiliser le « + »). Ainsi, vous pouvez spécifier une même *location* (registre ou mémoire) comme opérandes de sortie et d'entrée à la fois. La connexion entre les deux opérandes doit être décrite par une contrainte disant que les deux opérandes doivent occuper la même *location* à l'exécution de l'instruction correspondante. Notez qu'on peut utiliser la même expression C pour les deux opérandes, comme on peut utiliser deux expressions différentes. Pour illustrer, prenons l'exemple suivant :

```
1  __asm__( "addl %2, %1" : "=r" (a) : "0"
       (a), "r"(b));
```

La contrainte *"0"* dans le premier opérande d'entrée (d'indice 1) dit que l'expression C doit occuper le même registre général que celui spécifié dans l'opérande de sortie (d'indice 0).

Important : un nombre n'est autorisé comme contrainte que dans un opérande d'entrée et il doit se référer à un opérande de sortie.

Seul un nombre dans une contrainte peut garantir que deux opérandes vont occuper la même *location*. L'utilisation de la même expression dans les deux opérandes ne suffit pas. Ainsi, le résultat du code suivant n'est pas fiable :

```
1  __asm__( "addl %2, %1" : "=r" (a) : "r"
       (a), "r"(b));
```

En fait, Le compilateur peut choisir deux registres généraux différents pour stocker les deux opérandes 0 et 1.

4.5 Déclaration des modifications

Si une instruction modifie (explicitement ou implicitement) les valeurs d'un ou plusieurs registres, spécifiez ces registres dans la quatrième section de la construction *asm*. Les registres modifiés sont décrits dans des chaînes de caractères séparées par des virgules. Dans l'exemple suivant, la valeur du registre *EBX* est modifiée par l'instruction *movl*. Donc, on doit écrire son nom dans la section 4 de la construction *asm* :

```
1  #include<asm/unistd.h>                    /*
       __NR_write */
2  #include<unistd.h>                        /*
       STDOUT_FILENO */
3  #include<string.h>                        /*
       strlen() */
4
5  #define STDOUT STDOUT_FILENO
6
7  int main(void)
8  {
9   char * msg = "hello!\n";
10  int res;
11
12  __asm__("movl \t%2, %%ebx\n\t"
13          "int \t$0x80"
14
15          :"=a" (res)
16          :"0" (__NR_write), "I" (STDOUT)
                ,
17           "c" (msg), "d" (strlen(msg))
18          :"ebx");
19
20 return 0;
21 }
```

L'instruction assembleur *int* utilise, implicitement, les registres *EAX*, *EBX*, *ECX* et *EDX*. Le registre *ECX* contient l'adresse du premier caractère de la chaîne *msg*, et *EDX* contient sa taille. Le registre *EAX* est utilisé par l'instruction *int* comme opérandes d'entrée et de sortie à la fois. En tant qu'opérande d'entrée, il doit être initialisé avec la valeur numérique *__NR_write*. Ainsi, l'instruction *int* va générer l'appel système 4 (la fonction *write()* du fichier */usr/include/unistd.h*), pour afficher le message « hello ! » sur la console. La valeur de retour de l'appel système sera stockée dans le registre *EAX*, en tant qu'opérande de sortie.

Important : vous ne devez jamais écrire le nom d'un registre dans la section de déclaration des modifications, si ce registre fait partie d'un opérande d'entrée ou de sortie.

En fait, un tel registre est, à l'avance, déclaré modifié. Dans l'exemple ci-dessus, les registres *ECX ("c")*, *EDX ("d")* et *EAX ("=a")* sont utilisés pour stocker les valeurs des opérandes d'entrée et de sortie. Donc, on n'a pas listé leurs noms dans la section 4 de la construction *asm*.

En utilisant les informations sur la modification des registres, le compilateur détermine les valeurs qui doivent être sauvegardées dans la pile et restaurées après l'exécution du bloc *asm*.

4.5.a Modification de la mémoire : "memory"

Si vos instructions assembleur accèdent à la mémoire d'une manière imprévisible et arbitraire, ajoutez *"memory"* à la liste des modifications. Cela va renseigner à *GCC* de ne pas maintenir les valeurs (destinées à être chargées dans la mémoire) stockées dans des registres et de ne pas optimiser l'accès à la mémoire, durant l'exécution des instructions. D'autre part, si vous connaissez la taille de la mémoire accédée, ajoutez-la comme entrée, sinon utilisez *"memory"*.

4.5.b Modification du registre code condition : "cc"

Si vos instructions assembleur peuvent modifier le registre de code condition *cc* d'une manière inhabituelle, ajoutez *"cc"* à la liste de modifications. Avec l'architecture *x86*, la notation *cc* est utilisée par *GCC* pour représenter le registre des indicateurs *EFLAGS*.

Certaines instructions de l'architecture *x86*, telles que celles de décalage et de rotation logique ou arithmétique, peuvent altérer quelques bits du registre *EFLAGS* pendant leur exécution. Pour illustrer, soit l'exemple suivant :

```
1  __asm__( "shll $2, %1" : "=r" (res) : "r
       " (a) : "cc");
```

Dans l'exemple ci-dessus, l'instruction *shll* (*Logical Left Shift*) décale les bits du registre deux positions vers la gauche. Le dernier bit (de gauche) décalé est transféré dans le bit *CF* (*Carry Flag*). Ainsi, le registre code condition (*EFLAGS*) est toujours modifié avant qu'on puisse le tester ! Donc, *"cc"* doit être placé dans la section 4 pour renseigner le compilateur. Par contre, le bit *OF* (*Overflow Flag*) sera modifié automatiquement par le processeur comme résultat de l'exécution de *shll*, ce qui n'est pas pris en compte par le compilateur comme modification du registre de code condition *cc*.

Les instructions de test *TEST*, de comparaison *CMP*, ainsi que les instructions de contrôle des indicateurs comme *CLD* et *STD* n'ont pas d'opérandes de sortie (comme *shll*). Ainsi, ils ne sont pas considérés par le compilateur comme modificateurs de registre *cc*.

4.5.c Le modificateur « & »

À moins qu'un opérande de sortie contienne « & » dans sa contrainte, *GCC* peut le stocker dans le même registre alloué à un opérande d'entrée qui ne lui fait pas référence. C'est parce que le compilateur *GCC* suppose toujours que les instructions assembleur finissent l'utilisation des opérandes d'entrée avant que les sorties soient produites. Or, dans certains cas, cette supposition est fausse, ce qui peut provoquer des problèmes. Ainsi, le programme suivant n'est pas fiable :

```
1  __asm__("movl %1, %%eax\n\t"
2          "addl %2, %%eax\n\t"
3          "subl %1, %1"
4          :"=a" (res)
5          :"r" (a), "b" (b));
```

Si le compilateur alloue le registre *EAX* au premier opérande d'entrée, le résultat de l'addition (contenu du registre *EAX*) sera modifié avant son chargement dans *res*. C'est parce que la production de sortie *res* aura lieu juste après l'exécution de l'instruction *subl* ! Voici une portion du code-cible du programme ci-dessus :

```
1          movl         $3, %edx
2          movl         $10, %eax
3  #APP
4  # 9 "addition2.c" 1
5          movl          %eax, %eax
6          addl          %edx, %eax
7          subl          %eax, %eax
8  # 0 "" 2
9  #NO_APP
```

Seule l'utilisation du modificateur « & », dans la contrainte de l'opérande de sortie, va prévenir l'allocation du registre *EAX* à l'opérande d'entrée.

Dans l'exemple ci-dessus, l'utilisation de *"0"* comme contrainte du premier opérande d'entrée n'est pas correcte. C'est parce qu'il est utilisé par l'instruction *subl*, en tant qu'opérande d'entrée, avant la production de la sortie *res*. Par contre, dans l'exemple suivant, l'utilisation de la contrainte *"0"* est correcte :

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5   int x = 10;              /* EAX */
6   int mult = 3;
7   int res;                 /* EAX */
8
9   __asm__("movl %2, %%ebx\n\t"
10          "imul %%ebx"
11          :"=&a" (res)
12          :"0" (x), "r" (mult)
13          :"ebx");
14
15  printf("%d * %d = %d \n", x, mult, res)
        ;
16  return 0;
17 }
```

L'instruction *imul* utilise implicitement le registre *EAX* pour stocker le nombre à multiplier (comme entrée) et pour stocker le résultat de la multiplication. Ainsi, le premier opérande d'entrée est utilisé seulement avant la production de la sortie *res*. Voici un autre exemple montrant l'importance du modificateur « & » dans quelques cas :

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5   int x = 10;              /* EAX */
6   int divs = 3;
7   int quot;                /* EAX */
8   int reste;               /* EDX */
9
10  __asm__("subl %%edx, %%edx\n\t"
11          "movl %3, %%ebx\n\t"
12          "idivl %%ebx"
13
14          :"=a" (quot), "=&d" (reste)
15          :"0" (x), "r" (divs)
16          :"ebx");
17
18  printf("%d / %d = %d \n", x, divs, quot
        );
19  printf("%d %% %d = %d \n", x, divs,
        reste);
20  return 0;
21 }
```

L'instruction *idivl* utilise implicitement le registre *EAX* pour stocker le dividende (x) et le quotient ($quot$). Le registre *EDX* sera utilisé implicitement pour stocker le reste ($reste$) de la division. Ainsi, le registre *EDX* doit être indiqué dans la deuxième contrainte de sortie.

Supposons qu'on n'ait pas utilisé l'identificateur « & » dans la deuxième contrainte. Dans ce cas, le compilateur peut allouer le registre *EDX* pour stocker le diviseur ($divs$). Et par conséquent, l'exécution du programme peut provoquer une exception de type division par 0 !

Ainsi, lorsque le nom d'un registre est indiqué comme contrainte de sortie, il vaut mieux d'utiliser le modificateur « & ».

5 Problèmes d'optimisation

Durant l'optimisation, *GCC* tente de réordonner et de réécrire le code du programme, même en présence de la construction *asm*. Si les opérandes de sortie ne sont pas utilisés (la section 2 de *asm* est vide), ou si leurs valeurs ne sont pas modifiées par les instructions, l'optimiseur considère que les instructions n'ont pas un effet de bord dans le programme. Ainsi, la construction *asm* peut être supprimée. Pour bien comprendre le problème, on va étudier la portion de code suivante, prise du fichier *include/string.h* du

code source du noyau *Linux-0.01*) :

```
1  extern inline char *strcpy(char *dest,
       const char *src)
2  {
3
4   __asm__("cld\n"
5           "1: lodsb\n\t"
6           "stosb\n\t"
7           "testb %%al,%%al\n\t"
8           "jne 1b"
9           ::"S" (src), "D" (dest)
10          :"ax", "memory");
11  return dest;
12 }
```

La construction *asm* du code ci-dessus ne possède pas d'opérande de sortie. Pour le moment, on n'a pas besoin d'utiliser de sortie puisque les instructions *lodsb* et *stosb* utilisent directement la mémoire. Ainsi, on n'a pas de résultat à récupérer depuis un registre dans la mémoire à la fin de l'exécution du bloc *asm*. Mais l'optimiseur est parfois fou ! L'optimisation risquera de supprimer la construction *asm*. Et, par conséquent, la fonction *strcpy()* devient inutilisable.

Vous pouvez empêcher la construction *asm* d'être supprimée, par l'utilisation du mot-clé *volatile*, qui va indiquer au compilateur que les instructions ont un effet de bord important. *GCC* ne supprime jamais un *asm* volatile. Mais il peut le déplacer dans le code. Pour éviter ça, vous devez toujours spécifier tout opérande modifié dans le code assembleur, comme opérande de sortie. Ainsi :

```
1  extern inline char *strcpy(char *dest,
       const char *src)
2  {
3   int S, D, A;
4
5   __asm__ __volatile__(
6           "cld \n"
7           "1:lodsb \n\t"
8           "stosb \n\t"
9           "testb %%al, %%al \n\t"
10          "jne 1b"
11
12          :"=&S (S)", "=&D (D)", "=&a (A)
              "
13          :"0" (src),"1" (dest), "2" (0))
                ;
14  return dest;
15 }
```

Dans l'exemple ci-dessus, le contenu des registres *ESI*, *EDI* et *EAX* a été modifié. Ainsi, on les a spécifiés comme opérandes de sortie. Les variables *S*, *D* et *A* déclarées dans le code C sont utilisées juste pour créer une dépendance avec le bloc *asm*. Ainsi, l'optimiseur ne déplacera pas ce dernier.

6 Utilisation des macros

Si vous décidez d'utiliser des instructions assembleur dépendantes de l'architecture, il vaut mieux pour vous les encapsuler dans des macros et les placer dans un fichier d'en-tête. Cela peut vous aider à la maintenance de vos programmes. Ainsi, si vous décidez de porter un programme vers une autre architecture, vous n'aurez besoin de réécrire qu'un seul fichier. Dans l'exemple *max.c*, on a encapsulé la construction *asm* dans la macro *max(a,b)* définie dans le fichier d'en-tête *max.h* :

```
1  #ifndef MAX_H
2  #define MAX_H
3
4  #define max(a,b)                          \
5  ({                                        \
6   int __res, __x = (a), __y = (b);         \
7   __asm__ __volatile__(                    \
8           "cmpl %1, %2\n\t"                \
9           "jge 1f\n\t"                     \
10          "movl %1, %0\n\t"                \
11          "jmp 2f\n\t"                     \
12          "1: movl %2, %0\n\t"             \
13          "2:"                             \
14          :"=r" (__res)                    \
15          :"r" (__x), "r" (__y));          \
16  __res;                                   \
17 })
18
19 #endif
```

Notez que la dernière instruction d'une instruction composée, écrite entre « ( ) », doit être une expression suivie par « ; ». Elle sert à donner une valeur à l'instruction entière. Dans notre exemple, on veut récupérer la valeur de l'opérande de sortie dans le code C, pour l'afficher par exemple. Ainsi, on a écrit *__res ;* à la fin de l'instruction composée.

Dans la définition de la macro *max(a,b)*, les variables *__x* et *__y* sont utilisées pour s'assurer que la construction *asm* opère sur des valeurs entières. Une autre méthode pour faire en sorte que la construction *asm* opère sur le type de données correct est l'utilisation de forçage de type (*casting*) dans les entrées. Dans l'exemple ci-dessus, on peut forcer l'utilisation du type entier dans les opérandes d'entrée de la construction *asm* comme suit :

```
1  (::"r" ((int)a), "r" ((int)b);
```

Pour le même objectif, les constructions *asm* peuvent être encapsulées dans des fonctions en ligne. La section suivante en donne un exemple.

7 Un exemple détaillé

7.1 Opérations sur les chaînes de caractères avec l'architecture x86

En bref, les instructions *x86* de manipulation des chaînes des caractères, comme *lodsb* et *stosb*, utilisent implicitement les registres *ESI* et *EDI*. *ESI* doit contenir l'adresse, dans le segment *DS*, de la chaîne source, et *EDI* doit contenir celle, dans le segment *ES*, de la chaîne de destination. Le registre *EAX* est utilisé implicitement par ces instructions pour stocker temporairement les données traitées. L'instruction *cld* va mettre à 0 le bit *DF* (*Direction Flag*) du registre *EFLAGS*. Ainsi, *ESI* et *EDI* seront incrémentés par le processeur durant l'exécution des instructions. *A contrario*, l'instruction *std* est utilisée pour mettre à 1 le bit *DF*, et ainsi *ESI* et *EDI* seront décrémentés.

Le programme *string.c* utilise les deux fonctions en ligne *my_strcpy* et *my_strcmp* définies dans le fichier *string.h*. La première fonction va initialiser une chaîne de caractères ; la deuxième va comparer deux chaînes initialisées et retourner 0 si elles sont égales, sinon elle retourne -1 ou 1.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "string.h"
4
5  int main (void)
6  {
7   char * str1 = (char*)
8          malloc(sizeof(char*));
9   char * str2 = (char*)
10         malloc(sizeof(char*));
11
12  my_strcpy("fooo", str1);
13  my_strcpy("foo", str2);
14
15  if(my_strcmp(str1, str2)==0)
16         printf("Les deux chaînes \"%s\"
               et \"%s\" "\
17           "sont égales.\n", str1, str2);
18  else
19         printf("Les deux chaînes \"%s\"
               et \"%s\" "\
20           "ne sont pas égales.\n", str1,
                 str2);
21
22  free (str1);
23  free (str2);
24  return 0;
25 }
```

7.2 La fonction my_strcpy

```
1  static inline void
2  my_strcpy(char * src, char * dest)
3  {
4   int S, D, A;
5
6   __asm__ __volatile__ (
7          "cld\n\t"                    /* ESI++
                , EDI++ */
8          "1:\tlodsb\n\t"              /* MOVB
               DS:ESI, AL */
9          "stosb\n\t"                  /* MOVB
               AL, ES:EDI */
10         "testb\t%%al,%%al\n\t"       /* ZF=1
               si AL == 0 */
11         "jne\t1b"                    /* JMP
               si ZF == 1 */
12
13         :"=&S" (S),"=&D" (D), "=&a" (A)
14         :"0" (src),"1" (dest), "2" (0)
15         :"memory");
16 }
```

Dans le code assembleur, on a utilisé l'instruction *cld*. Donc, *ESI* et *EDI* doivent contenir les adresses *src* et *dest* du premier caractère de chaque chaîne.

En outre, les instructions *lodsb* et *stosb* accèdent à la mémoire (*src* et *dest*) d'une manière imprévisible. En effet, on ne connaît pas à l'avance le nombre d'octets (caractères) à copier. Donc, on a utilisé *"memory"*.

D'autre part, les trois registres *ESI*, *EDI* et *EAX* seront modifiés par les instructions assembleur, donc on les a spécifiés comme opérandes de sortie. Ainsi, on doit utiliser le modificateur « & » pour prévenir l'allocation de ces registres à une entrée non correcte. La connexion entre les entrées et les sorties aura lieu avec l'utilisation des contraintes "0", "1" et "2". Cela est autorisé parce que les sorties *S*, *D* et *A* seront produites après la consommation des entrées par les instructions. Elles sont utilisées juste pour créer une dépendance entre le code C et le bloc *asm* !

L'utilisation du mot clé *volatile* et des opérandes de sortie va prévenir la suppression ou le déplacement de la construction *asm* pendant l'optimisation.

7.3 La fonction my_strcmp

La fonction *my_strcmp* est aussi extraite du fichier *include/string.h* du code source du noyau *Linux-0.01*. Son fonctionnement est similaire à celui de la fonction *strcmp* de la bibliothèque C standard, définie dans le fichier */usr/include/string.h*.

```
1  static inline int
2  my_strcmp(const char * str1,const char *
       str2)
3  {
4   int S, D, __res;
5
6   __asm__ __volatile__(
7          "cld\n"                      /* ESI++,
                EDI++ */
8          "1:\tlodsb\n\t"              /* MOV DS
               :ESI, AL */
9          "scasb\n\t"                  /* SUB ES
               :ESI, AL */
10         "jne 2f\n\t"                 /* JMP si
               ZF == 0 (ES:EDI != AL) */
11         "testb %%al,%%al\n\t"        /* ZF=1
               si AL == 0 */
12         "jne 1b\n\t"                 /* JMP si
               ZF == 0 => il ya encore des
```

```
                caractères */
13                                  /* pour
                                        comparer
                                         */
14          "xorl %%eax,%%eax\n\t" /* (str1
                == str2) => __res = 0 */
15          "jmp 3f\n"              /* on a
                terminé ! */
16          "2:\tmovl $1,%%eax\n\t"
17          "jl 3f\n\t"             /* (str1
                != str2) et (str1[i] > str2[
                i]) => __res = 1 */
18          "negl %%eax\n"          /* (str1
                != str2) et (str1[i] < str2[
                i]) => __res = -1 */
19          "3:"
20
21          :"=&D" (S), "=&S" (S),"=&a" (
                __res)
22          :"0" (str1),"1" (str2), "2" (0)
23          :"memory");
24
25   return __res;
26  }
```

7.4 La compilation avec GCC

Voici le *makefile* utilisé pour compiler le programme *string.c* :

```
1   CFLAGS = -O2 -fomit-frame-pointer -W -
        Wall
2
3   string: string.o
4           cc string.o -o string
5
6   string.o: string.c string.h
7           cc -c $(CFLAGS) string.c -o
                string.o
8
9   clean:
10          rm -rfv string.o
```

O2 est le niveau d'optimisation recommandé. Le compilateur va essayer d'augmenter les performances sans compromettre la taille et sans prendre trop de temps en compilation. Ce niveau d'optimisation permet de produire un code rapide. L'option *-fomit-frame-pointer* permet aussi de produire un code rapide et de taille réduite. Consultez le manuel de *GCC* pour avoir accès à encore plus d'informations.

*Retrouvez l'article d'**Issam Abdallah** en ligne : lien 5*

Eclipse

Les dernières news Eclipse

La nouvelle version de l'environnement Eclipse est disponible, Eclipse Luna apporte le support natif de Java 8

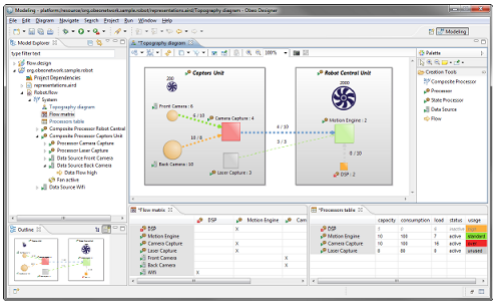
L'environnement de développement open source Eclipse évolue pour répondre aux besoins des développeurs. Eclipse 4.4 est disponible en téléchargement. Baptisé Luna (déesse de la Lune dans la mythologie romaine), l'EDI promet d'illuminer le quotidien des développeurs avec son lot de nouveautés.

Le fait marquant de cette nouvelle version est, sans aucun doute, le support natif de Java 8. En effet, même si Eclipse Kepler supportait Java 8 dès sa sortie, il fallait lui adjoindre un patch afin que les développeurs Java puissent profiter des Lambdas.

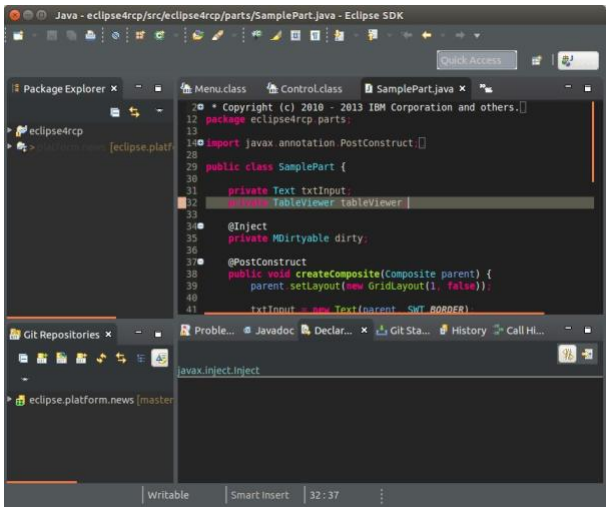
Des informations supplémentaires sur les nouveautés de cette version sont disponibles à cette adresse : https ://projects.eclipse.org/releases/luna. Eclipse Luna se compose de 76 projets, pour un total d'environ 10 millions de nouvelles lignes de code. 339 contributeurs à travers le monde ont participé à cet effort. Huit projets ont rejoint le « simulatenous release train » : EMF Client Platform, EMF Store, Sirius, BPMN2 et BPMN2 Modeler Project, Paho QVTd et XWT. L'ensemble des projets disponibles dans cette version est disponible à cette adresse : https ://projects.eclipse.org/releases/luna/details

Sans être exhaustifs, les principales nouveautés autres que le support de Java 8 sont :

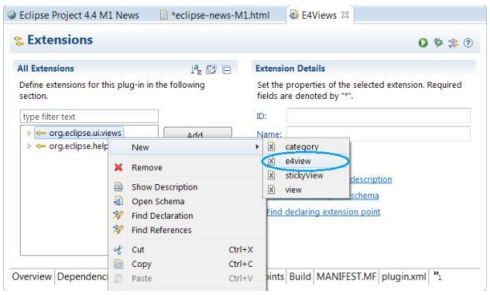
— le projet Sirius, qui permet de construire de manière simple et totalement nouvelle des éditeurs, basé sur le framework de métamodélisation EMF et sa couche graphique spécifique GMF. Avec Eclipse Sirus, le développement d'éditeurs devient déclaratif ;



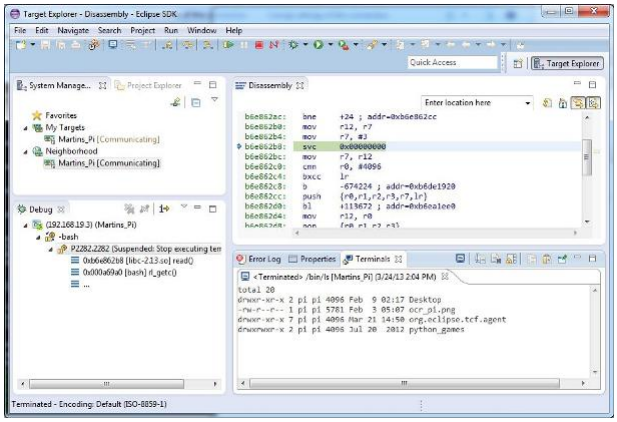
— un nouveau thème graphique "sombre" très proche de ce qui a été proposé par les dernières versions de l'environnement de développement IntelliJ ;



— la possibilité de créer des vues de type "E4" depuis un plugin. La nouvelle API E4 ne sera donc plus réservée aux applications Eclipse RCP ;



— un client de type terminal sous Windows, Linux et MAC via le projet TCF ;



— une amélioration sensible du Workbench, qui permet notamment de pouvoir séparer un même éditeur en 2 parties, soit horizontalement, soit verticalement ;

— l'éditeur graphique d'EMF (Editor Diagram) a été entièrement rebâti en se basant sur le projet Sirus ;

— une amélioration de la bibliothèque EGIT, l'implémentation Java de GIT. L'outil graphique de fusion a également été amélioré.

Pour télécharger cette nouvelle version, rendez-vous sur la page de téléchargement d'Eclipse : lien 6

Cette courte vidéo revient sur les principales nouveautés : lien 7

Et vous ?

— Que pensez-vous de cette nouvelle version ? Et de ces nouvelles fonctionnalités ?

— Avez-vous déjà essayé cette nouvelle version ?

— Allez-vous migrer ?

*Commentez la news de **Mickael Baron** en ligne : lien 8*

Eclipse Orion 6 débarque avec un support amélioré de JavaScript, quatre mois après la sortie d'Orion 5

Près de quatre mois après la sortie de la version 5, l'équipe chargée du développement d'Eclipse Orion -la solution de développement web dans le cloud- est déjà prête à annoncer la sortie de la version 6.

Alors que la précédente version mettait l'accent sur une interface graphique plus soignée (lien 9), en plus de différentes améliorations (validateur syntaxique pour JavaScript, support amélioré de Gerrit, assistant de contenu pour Node.js), cette nouvelle version sera axée principalement sur JavaScript et l'amélioration de son support.

Ken Walker, chef du projet Eclipse Orion a ainsi déclaré : *« Le support initial pour les conteneurs Docker.io pour les espaces de travail dédiés aux développeurs a été publié, notre équipe travaille avec l'équipe de la fondation Eclipse afin de les rendre accessibles sur OrionHub. Grâce à Docker.io un développeur sera en mesure d'exécuter des outils standards de Node.js dans leur espace de travail Orion. Si vous êtes un développeur Node.js, alors l'utilisation de Docker.io facilitera vos développements dans le cloud. ».*

« Le support de la validation de JavaScript qui est basé sur ESlint a subi une refonte avec de nouvelles règles, un moteur mis à jour et une plus grande flexibilité pour l'utilisateur. Le validateur scanne les fichiers JavaScript (ainsi que les balises HTML script) et signale les erreurs/avertissements sur la marge de l'éditeur. Un survol de souris permet d'obtenir plus de détails sur la nature de l'erreur ou de l'avertissement. » C'est ce qu'a déclaré un autre membre du projet, Curtis Windatt.

Dans les faits, cette nouvelle version facilite la génération de la documentation JavaScript à travers l'introduction de nouvelles commandes, analyse le code JavaScript à la recherche d'erreurs éventuelles grâce aux règles du validateur ESLint et met à jour le parseur, de quoi améliorer le quotidien des développeurs JavaScript sous Eclipse Orion.

Source : Annonce d'Orion 6.0 : lien 10

Et vous ? Qu'en pensez-vous ?

*Commentez la news d'**Arslan Hamza** en ligne : lien 11*

Les derniers tutoriels et articles

Tutoriel : créer un éditeur de diagramme facilement avec Eclipse Sirius

Qui n'a pas rêvé d'avoir simplement et rapidement un éditeur graphique en quelques minutes ? Qui n'a pas rêvé de pouvoir effectuer des changements à la volée sur cet éditeur afin de le construire avec son client « en live » ? Vous en rêviez, Sirius le fait. Rendre la création de ces éditeurs simple, rapide et à la portée de tous est le défi que se fixe le projet Eclipse Sirius. Dans ce tutoriel nous découvrirons ensemble comment créer son premier éditeur de type « diagramme » grâce à cette technologie. Les sources de cet exemple sont disponibles à l'adresse suivante : FTP (lien 12) ou HTTP (lien 13).

1 Introduction

1.1 Concept

La création d'éditeurs graphiques (éditeurs UML, représentation de processus, éditeurs de concepts métier, etc.) est difficile à intégrer dans un cycle itératif court avec le client, car coûteuse en temps et nécessitant souvent l'apprentissage de technologies ou frameworks complexes. Le projet Eclipse Sirius se propose de faciliter la création d'éditeurs en se basant sur la représentation d'un DSL au sein d'un modèle EMF (Eclipse Modeling Framework). Un DSL, Domain Specific Language, est un langage dédié à un domaine métier avec ses mots-clés et son formalisme. Ainsi, une recette de cuisine avec des ingrédients et une série d'actions est un DSL. Les DSL ne sont donc pas limités à l'informatique. Dans Sirius, à partir d'un modèle EMF, le développeur va pouvoir spécifier un éditeur de manière complètement déclarative pour le modèle et observer le résultat en temps réel, sans génération de code.

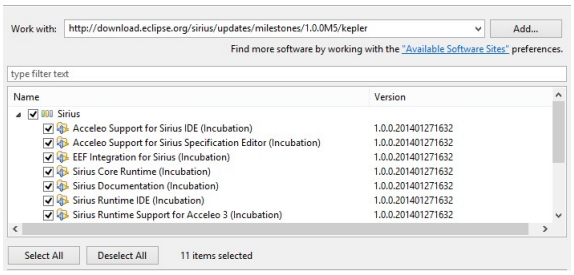
Dans ce tutoriel, nous allons détailler les différentes étapes depuis la création du modèle jusqu'à la spécification de l'éditeur avec Sirius, en expliquant les principaux concepts associés. Nous ne reviendrons pas sur les concepts d'EMF qui sont expliqués dans ce tutoriel : lien 14.

1.2 Un peu d'histoire

Le projet Sirius a été développé il y a quelques années par la société Obeo, en partenariat avec Thales. Les mécanismes servent notamment de base à l'application propriétaire Obeo Designer. Le projet passe maintenant en Open Source et sera déployé dans sa version 1.0 avec Eclipse Luna dans le « Simultaneous Release Train ». Malgré la numérotation « 1.0 », le projet possède déjà de nombreuses années derrière lui, ce qui garantit donc sa stabilité.

1.3 Prérequis et installation

Pour commencer, il vous faut une installation d'Eclipse Juno, Kepler ou supérieure. Votre distribution doit comporter les outils de création de modèles EMF. Pour simplifier l'installation, vous pouvez télécharger la version packagée « Eclipse Modeling Tools » sur le site de téléchargement officiel. On peut alors installer Sirius. Actuellement, comme mentionné précédemment, Sirius n'est pas disponible sur les repositorys, officiels. Vous trouverez les liens de téléchargement associés sur la page du projet : lien 15. Pour ce tutoriel, nous nous baserons sur la version 1.0.0M5 d'Eclipse Kepler. Sélectionnez tous les composants de la catégorie « Sirius ».



Packages Sirius

Dans la première partie de ce tutoriel, nous verrons comment mettre en place le modèle EMF. Dans les parties suivantes, nous mettrons en place notre éditeur avec Sirius.

2 Création du modèle EMF

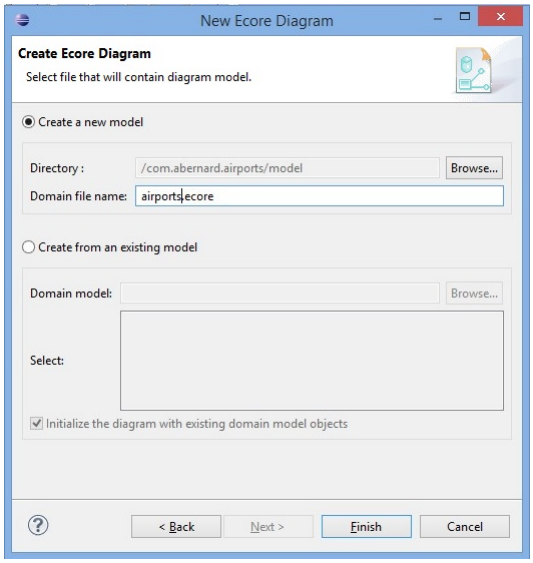
2.1 Notre DSL

Nous allons, dans cet article, créer un DSL très simple qui nous permettra de mettre en œuvre cer-

tains mécanismes clés de Sirius. Notre DSL nous servira à représenter les liaisons aériennes qui existent entre différents aéroports. Le modèle contiendra donc un certain nombre d'aéroports définis par un nom, une ville et un pays. Ces aéroports ont un ensemble de portes identifiées par un numéro. Ces portes peuvent référencer une porte d'un autre aéroport, représentant ainsi une liaison entre deux aéroports.

2.2 Création du modèle

Créez d'abord un nouveau projet de type « Empty EMF Project », puis créez un fichier « Ecore Diagram » au sein du dossier « model » du nouveau projet , et enfin, créez un nouveau modèle appelé « airports.ecore ».



Création du diagramme Ecore

Créez ensuite chacun des éléments du modèle pour obtenir le diagramme suivant :

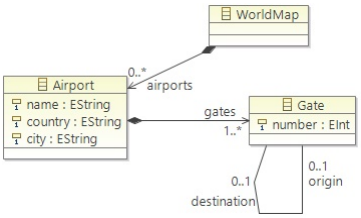
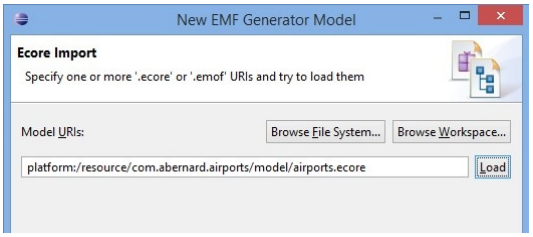


Diagramme du modèle EMF

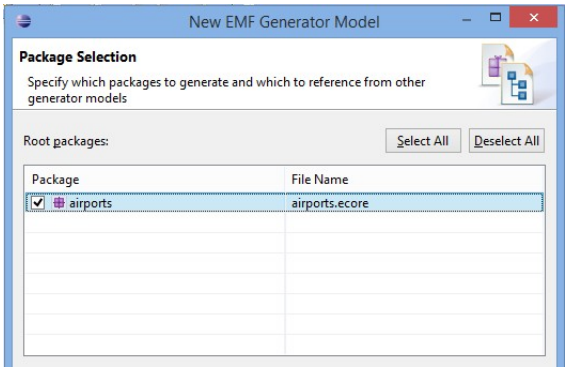
2.3 Génération

Créez un fichier « .genmodel » à partir de l'assistant « EMF Generator Model » pour notre fichier « airports.ecore » : sur la deuxième page « Select Model importer », sélectionnez « Ecore model », puis sur la page suivante, sélectionnez notre modèle « .ecore » et cliquez sur « Load » :



Chargement du modèle Ecore

Sélectionnez ensuite le package « airports » pour la génération :



Sélection du package Ecore

Une fois le fichier « .genmodel » créé, ouvrez-le et lancez la génération du modèle, du plugin « edit » et du plugin « editor ». Nous obtenons donc trois plugins, « com.abernard.airports », « com.abernard.airports.edit » et « com.abernard.airports.editor ».

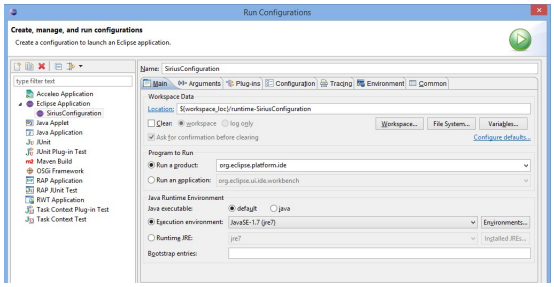
3 Création de l'éditeur

3.1 Fonctionnement de Sirius

La grande force de Sirius est de permettre la spécification des éditeurs de manière totalement graphique, sans avoir à écrire de code. Parallèlement à la spécification, le rendu peut être observé en temps réel à partir de l'instanciation du modèle que l'on veut représenter. De ce fait, nous allons travailler dans une instance d'Eclipse qui embarque les plugins que nous avons générés précédemment, de manière à pouvoir créer un exemple de notre modèle grâce aux éditeurs par défaut. Sirius est aussi une énorme avancée dans la construction d'éditeur de diagrammes dans le sens où la seule alternative était GMF (Graphical Modeling Framework), extrêmement complexe à prendre en main et peu intuitif.

La première étape consiste à lancer un Eclipse avec nos plugins. Pour cela, ouvrez d'abord le pan-

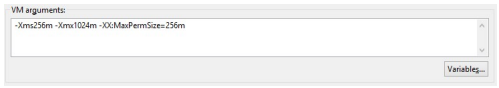
neau « Run configurations » via le menu « Run > Run configurations », puis créez une nouvelle configuration de type « Eclipse Application ».



Création de la configuration

Puis, ajoutez la ligne suivante dans la zone « VM arguments » de l'onglet « Arguments » :

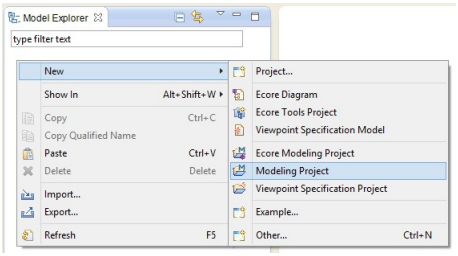
```
1 -Xms256m -Xmx1024m -XX:MaxPermSize=256m
```



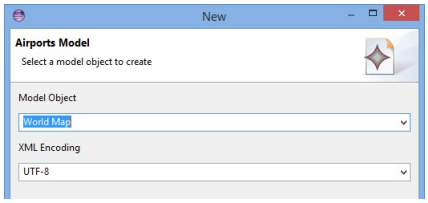
Lancez ensuite l'application. **C'est uniquement dans cette nouvelle application que nous travaillerons désormais**. En effet, Sirius se base sur les plugins EMF que nous avons créés et intégrés dans la nouvelle instance d'Eclipse pour la spécification et le test de l'éditeur.

3.2 Instanciation du modèle

Dans l'application, ouvrez la perspective « Sirius ». Cette dernière ouvre une vue appelée « Model Explorer ». Créez ensuite un projet de type « Modeling Project » appelé « com.abernard.airports.example ».

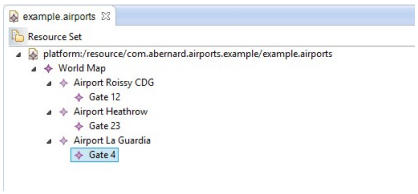


Dans le projet créé, on peut identifier un fichier appelé « representations.aird ». Ce dernier permettra à Sirius de stocker l'agencement des éléments dans l'éditeur que l'on va créer, de manière indépendante du modèle lui-même. Notez qu'il existe un seul fichier « .aird » par « Modeling Project », mais chaque fichier peut stocker plusieurs représentations. Créez ensuite une instance du modèle grâce au menu « New > Example EMF Model Creation Wizards > Airports model ». Donnez un nom à votre modèle, puis sélectionnez comme objet racine, l'objet « World Map ».



Création du modèle

Créez ensuite les éléments du modèle à votre convenance, mais avec au moins deux aéroports contenant chacun une porte, les portes sont reliées entre elles. Par exemple, voilà l'instance dont je vais me servir dans le reste de cet article.

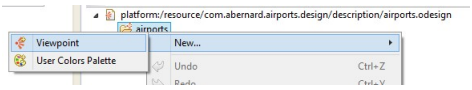


Exemple d'instance

3.3 Initialisation de l'éditeur

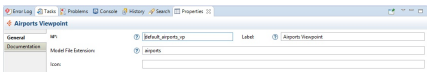
La création des éditeurs de Sirius se fait au travers des projets que l'on nomme « Viewpoint Specification Project ». Via le menu « New > Project > Viewpoint Specification Project », créez un projet « com.abernard.airports.design ». Afin de pouvoir le manipuler plus facilement lorsque nous aurons spécifié l'éditeur dans notre workspace de base, je vous conseille de créer le projet dans votre workspace de développement plutôt que celui de runtime. Sur la deuxième page de l'assistant, donnez un nom au fichier « .odesign » ; ce dernier contient la spécification de notre éditeur et c'est l'unique fichier que nous allons modifier.

Le fichier s'ouvre, affichant un unique élément « airports ». La première étape consiste à créer un « Viewpoint ». Comme son nom l'indique, un viewpoint sert à définir une manière de représenter un modèle. Cela permet de proposer différents points de vue, selon par exemple le niveau des utilisateurs. Avec un clic droit sur l'élément « airports », créez un viewpoint.



Dans la vue « Properties », trois champs sont à remplir : un champ « ID », un champ « Label » et un champ « Model File Extension ». Le champ « ID » est présent sur tous les éléments qui seront définis dans le fichier « odesign » et est obligatoire. D'une version à une autre, si vous modifiez l'ID de vos éléments, la compatibilité ne sera plus assurée avec les fichiers « representations.aird » existants. Le champ

« Label » est là pour donner un nom plus compréhensible aux éléments et peut être changé sans perturber les représentations existantes. Il est aussi présent sur la plupart des éléments et sera souvent le texte affiché aux utilisateurs. Dans le champ « Model File Extension », mettez l'extension des fichiers de modèle que vous voulez associer à votre viewpoint.



Nous devons associer ensuite une représentation à notre viewpoint afin d'indiquer à Sirius quel type d'éditeur nous voulons créer. Il est possible de créer des éditeurs sous forme d'arbre, de tableau, de diagramme ou encore de diagramme de séquence.

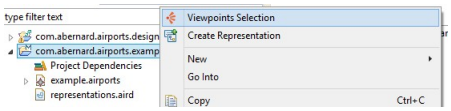
Dans cet exemple, nous allons créer un diagramme.

Faites un clic droit sur le viewpoint puis sélectionnez « New Representation > Diagram Description ». Dans la vue « Properties », renseignez l'ID, le label et le champ « Domain Class ». Ce champ indique l'objet de base représenté par le diagramme. Par exemple, pour un diagramme de classe, l'élément de base serait sans doute le package ; dans notre cas, c'est l'objet « WorldMap ». Notez que dans ce champ, vous pouvez utiliser l'auto-complétion.

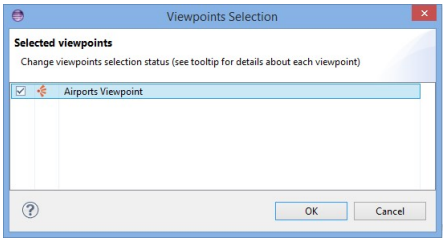
À partir de maintenant, il est possible de visualiser l'éditeur en temps réel au fur et à mesure que nous le spécifions.

3.4 Visualisation en temps réel

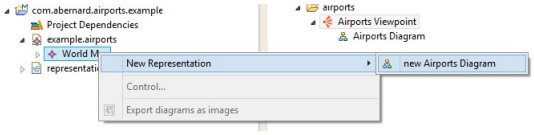
Pour lancer la visualisation, faites un clic droit sur le projet qui contient le modèle et sélectionnez « Viewpoints Selection ».



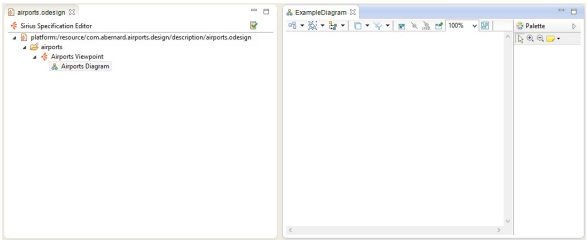
Dans la fenêtre qui s'affiche, sélectionnez le Viewpoint que nous venons de créer.



Enfin, dépliez le fichier « example.airports » et faites un clic droit sur l'objet « WorldMap » de notre exemple puis sélectionnez « New Representation » puis « new Airports Diagram » et donnez un nom à la représentation.

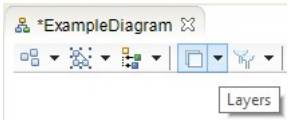


Le diagramme s'ouvre. Evidemment il est vide puisque nous n'avions défini aucune représentation pour les éléments de notre modèle. Nous pouvons le laisser ouvert et l'affichage s'adaptera à chaque modification que nous ferons dans le fichier « .odesign », en temps réel.



4 Spécification des éléments

La première étape consiste à spécifier un « layer ». Le layer permettra à l'utilisateur d'afficher ou non certains éléments à la volée grâce au bouton idoine dans la barre d'outils de l'éditeur généré.

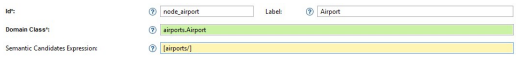


Sélectionnez l'élément « Airport Diagram » et dans le menu contextuel, sélectionnez « New Diagram Element > Default Layer Layer ». Comme d'habitude, donnez à cet élément un ID et un label.

4.1 Spécification des nœuds

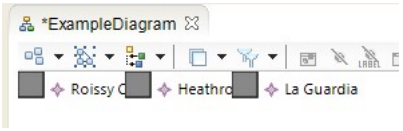
Nous devons ensuite définir la manière de représenter les éléments de notre modèle. Pour ce faire, vous allez créer des représentations pour les aéroports sous forme de rectangles, puis représenter les différentes portes sur les pourtours de ces rectangles. Pour commencer, grâce au menu contextuel sur le layer créé précédemment, vous créez un nouveau « Node Mapping » (« New Diagram Element » > « Node Mapping »). Donnez-lui un ID, un label, puis définissez dans le champ « Domain Class » la classe du modèle que ce nœud représentera ; dans notre cas, cela correspondra à la classe « Airport ». Le

champ « Semantic candidate » vous permet d'indiquer l'expression permettant d'accéder aux éléments à afficher. Cette expression est optionnelle mais si elle n'est pas remplie, tous les objets du modèle correspondant à l'instance seront affichés. Dans l'exemple ci-dessous, l'expression est indiquée mais donne exactement le même résultat que si le champ n'était pas rempli.



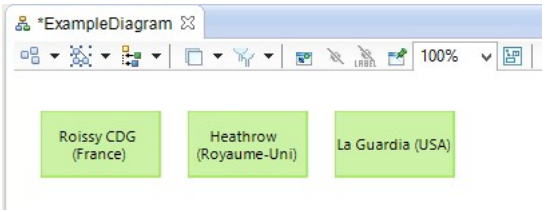
Remarquez la manière dont les expressions sont écrites, avec le contenu entre crochets : [. . . /]. Cette syntaxe correspond au langage Acceleo qui permet à Sirius d'interpréter les expressions sur des objets EMF. Pour plus d'informations, reportez-vous au site officiel : lien 16.

Il faut ensuite définir la représentation du nœud. Pour cela, faites un clic droit sur le « Node mapping » et sélectionnez « New Style > Square Description ». Enregistrez le fichier « .odesign » et observez l'éditeur que nous avions ouvert en regard : il s'est mis à jour et affiche les différents aéroports de notre modèle.

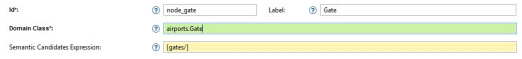


La vue « Propriétés » vous permet de définir le mode d'affichage des nœuds. L'onglet « Label » modifie le mode d'affichage du nom de l'élément : si on doit afficher l'icône, la position du nom et l'expression associée au nom.

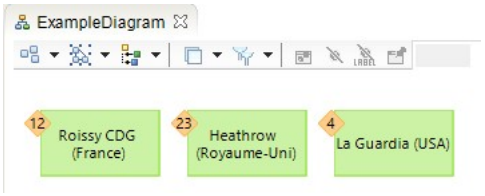
L'onglet « Color » règle les couleurs du nœud. Dans l'onglet « Advanced », on peut modifier la taille du nœud et la façon dont elle est calculée. À chaque modification d'une propriété, on peut observer les changements en temps réel dans l'éditeur.



Maintenant que nous avons défini la représentation des aéroports, nous pouvons créer, en bordure des éléments, les représentations de chacune des portes. Cela se fait par un clic droit sur l'élément « Node Mapping » des aéroports via « New Diagram Element > Bordered Node Mapping ».

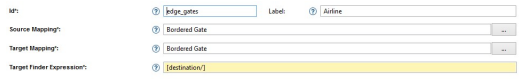


Puis, comme précédemment, nous pouvons régler les propriétés pour afficher ces éléments, par exemple via « New Style > Lozenge Style Description ». Dans ce cas, l'expression pour définir le label des éléments est « [number/] ». Encore une fois, l'éditeur se met à jour dès la sauvegarde du fichier.

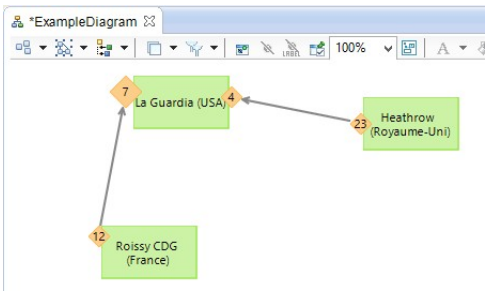


4.2 Spécification des relations

En plus des nœuds qui représentent les objets de notre modèle, Sirius permet de définir la représentation des relations entre les éléments. Pour cela, deux possibilités s'offrent à nous : soit un lien basé sur une relation entre deux objets (association, « containment ».. .), soit une relation basée sur un objet explicite. Dans notre cas il s'agit d'une relation entre deux objets de type « Gate » qui peut être créée via « New Diagram Element > Relation Based Edge ». L'élément est créé avec un style défini et la page de propriétés permet de définir les critères d'affichage de la relation entre deux éléments. Le champ « Source Mapping » donne le type d'objet source, le champ « Target Mapping » donne le type d'objet cible de la relation et le champ « Target finder expression » donne l'expression utilisée pour trouver l'objet cible depuis l'objet source.



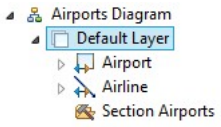
De nouveau, l'éditeur se met à jour et affiche les relations entre les différents éléments de notre modèle.



L'élément « Edge Style Description » permet de modifier la manière dont sont représentées les relations : on peut modifier, par exemple, les connecteurs ou la couleur des liens.

5 Création de la palette

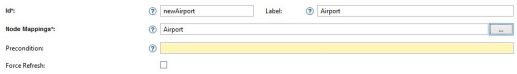
Les éléments que nous avons mis en place précédemment permettent de visualiser notre modèle dans un diagramme. Cependant, l'intérêt d'un éditeur reste tout de même de pouvoir éditer le modèle ! Nous allons dans cette section mettre en place la palette d'outils. Cette dernière peut servir à créer deux types d'éléments : d'une part les éléments visibles dans la palette du côté de l'éditeur et d'autre part, les mécanismes qui permettent de modifier les éléments à la volée, comme par exemple le label des éléments. Au niveau du layer de la définition de notre éditeur, créez une « Section » grâce au menu contextuel « New Tool > Section ». Comme d'habitude, donnez à cet élément un label et un ID.



5.1 Outils de création d'éléments

5.1.a Création des nœuds

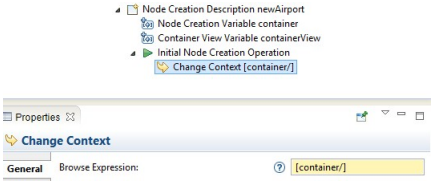
Pour plus de clarté, nous allons grouper dans des sous-sections les outils « visibles » et les outils de manipulation d'éléments. Créez donc une nouvelle section dans celle que nous venons de créer pour les outils de la palette « Palette tools ». Grâce au menu contextuel, sélectionnez « New Element Creation > Node creation Description ». Sur la page de propriétés qui s'affiche, donnez à cet outil un ID et un label, et définissez ensuite quel nœoeud cet outil va créer.



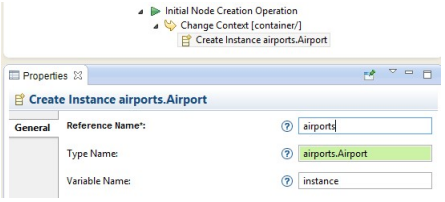
Il faut ensuite indiquer à Sirius comment instancier l'élément et où le placer au sein du modèle. Pour cela, les opérations doivent être décrites dans le nœud « Initial Node Creation Operation » de l'outil. La première chose à faire est de se placer dans le « contexte » adéquat via le menu contextuel « New Operation > Change Context » et définir l'expression afin de se placer dans l'objet « container ». Notez que Sirius, à cet emplacement, vous donne un accès à deux variables qui sont :

— « container » qui correspond à l'élément parent de la création de l'objet au sein de votre modèle (dans notre exemple, c'est directement l'objet « WorldMap ») ;

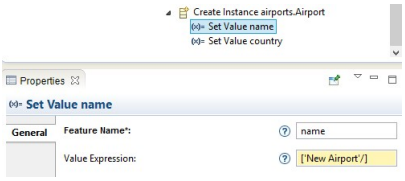
— et « containerView » qui est l'objet de l'élément « container » au sein du diagramme Sirius.



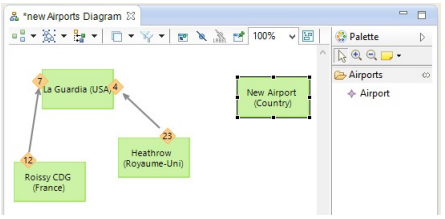
Créez ensuite une opération de type « Create Instance », puis définissez un nom de variable pour la nouvelle instance, la classe du modèle qui va être instanciée et l'attribut du container qui va faire le lien vers cette nouvelle instance : dans notre cas il s'agit de la liste d'aéroports de la classe « WorldMap », nommée « airports ».



Puis on peut définir des opération de type « Set Value » afin de définir des valeurs par défaut pour les attributs de l'instance que l'on vient de créer, par exemple le nom de l'aéroport ou le pays.



On peut constater que l'outil que nous venons de créer est apparu dans la palette et si l'on crée un nœud sur le diagramme, une nouvelle instance de l'objet « Airport » est créée dans notre modèle avec le nom et le pays par défaut que nous avons défini.

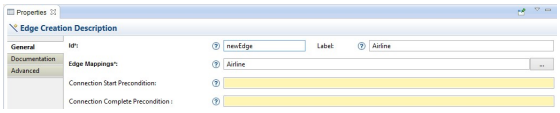



Notez que vous pouvez à tout moment effectuer une validation de votre fichier *.odesign grâce à l'icône « Validate Model ».

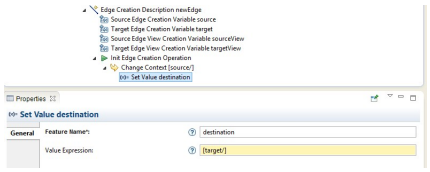
De la même manière, vous pouvez définir l'élément de création des objets « Gate » au sein de la palette.

5.1.b Création des connecteurs

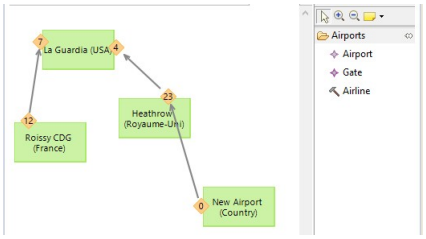
Pour définir l'outil de création des liens, faites un clic droit sur la section « Palette » puis choisissez « New Element Creation > Edge Creation Description ». Dans les propriétés, donnez un ID, un label et définissez quel élément du diagramme sera créé.



Comme pour la création des nœuds, Sirius vous donne directement accès à différentes variables représentant la source et la cible dans l'éditeur ainsi que leur correspondance au sein du modèle. Puis, comme pour les nœuds, il s'agit de définir l'enchaînement d'actions qui vont mener à la modification du modèle. La première étape consiste d'abord à se placer dans le contexte « [source/] » grâce à une opération « Change Context ». Puis, grâce à une opération « Set value », on définit la propriété « destination » de l'élément du modèle à « [target/] ».



Après sauvegarde, vous pouvez directement utiliser votre palette pour créer de nouveaux liens entre vos aéroports.

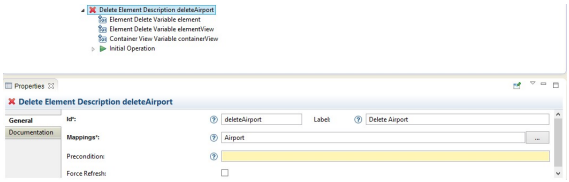


5.2 Outils de modification d'éléments

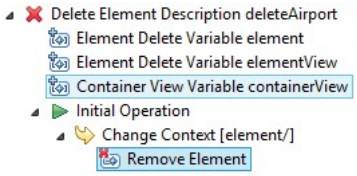
En plus des outils de création d'éléments, Sirius vous permet de définir les différentes actions sur votre diagramme, qui n'apparaissent pas dans la palette, mais permettent d'apporter des modifications à votre modèle. Nous ne verrons dans ce tutoriel que certains de ces outils. Commencez par créer une nouvelle section « Tools » au sein de la section principale « Airports ».

5.2.a Suppression des éléments

Si vous sélectionnez un élément et que vous appuyez sur la touche « Suppr », vous remarquerez que rien ne se passe. Ce comportement est défini grâce à « New Element Edition > Delete Element Description ». Comme d'habitude, donnez un ID et un label à votre action et définissez quels éléments du diagramme vont pouvoir être supprimés avec cette action.



Il est recommandé de se placer dans le contexte de l'élément via une opération « Change Context »et de supprimer l'élément avec une opération « Remove element ».



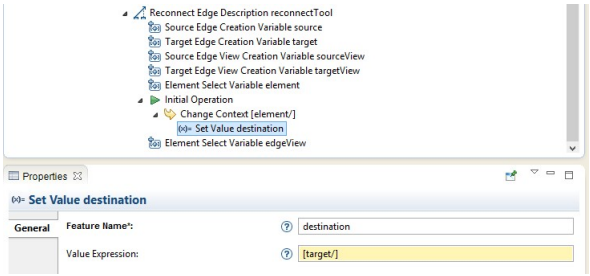
Vous pouvez ensuite tester dans votre éditeur qu'un appui de la touche « Suppr » lorsqu'un aéroport est sélectionné supprime bien l'élément du modèle.

5.2.b Modifier les liens entre éléments

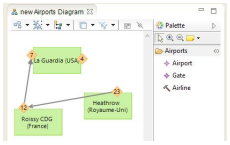
Une autre opération courante dans un éditeur de type diagramme est de pouvoir reconfigurer les connexions entre les différents éléments du diagramme. Là aussi, Sirius nous propose d'ajouter cette action prédéfinie à notre palette via « New Element Edition > Reconnect Edge Description ». Donnez un ID et un label à cet élément. Vous pouvez aussi définir quel type d'opération vous voulez gérer : soit « Reconnect Target » pour seulement autoriser la reconnexion de la cible de la connexion, soit « Reconnect Source » pour la reconnexion de la source uniquement, soit « Reconnect Both » pour gérer les deux cas de figure. Dans notre cas, nous nous intéresserons à « Reconnect Target » uniquement. Enfin, définissez sur quelle connexion cette opération s'applique dans le champ « Mappings ». Dans cette action, Sirius donne accès à trois variables :

- « source » : l'instance du modèle qui va être déconnecté ;
- « target » : l'instance du modèle qui va être connecté ;
- « element » : l'instance du modèle derrière la connexion elle-même.

Il faut ensuite définir la suite d'opérations à accomplir pour effectuer la reconnexion. Tout d'abord, on se place dans le contexte de la variable « element ». Puis on effectue une opération « set » sur la feature « destination » du modèle afin d'affecter la nouvelle destination à notre liaison aérienne : la variable « target ».

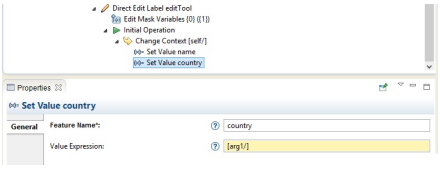


On peut alors reconnecter la destination des connexions entre éléments.

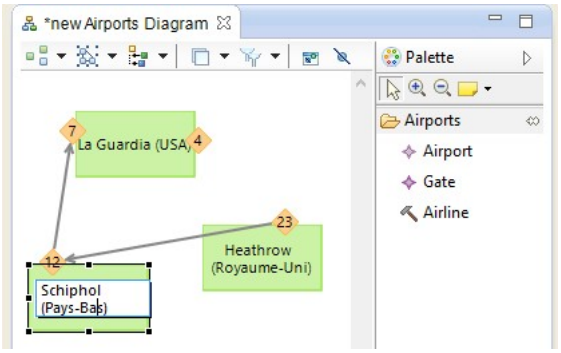


5.2.c Renommer les éléments

Enfin, Sirius permet de redéfinir à la volée le nom des éléments grâce à l'opération « New Element Edition > Direct Edit Label ». Pour effectuer cette opération, il faut définir sur quel élément graphique elle va s'appliquer (dans notre exemple le nœud « Airport ») en plus des habituels ID et label. Via une opération « Change context », on se place ensuite dans la variable « self » qui représente, dans Sirius, l'instance du modèle sous-jacente à l'élément graphique sélectionné. On peut alors définir une action « Set value » afin de redéfinir le nom de l'aéroport (la feature « name ») avec la valeur donnée par l'utilisateur, « arg0 ». La propriété « Edit Mask Variables » permet de définir une syntaxe pour l'édition des labels permettant d'accéder à plusieurs variables distinctes au sein de la chaîne de caractères entrée par l'utilisateur. Par exemple, nous indiquons dans notre cas le masque « 0 (1) » ; ce qui nous permet de récupérer dans la variable « arg0 » le nom de l'aéroport et dans la variable « arg1 » le pays.



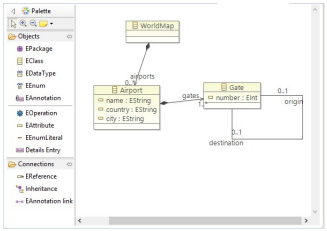
L'utilisateur peut alors, en un clic, modifier le nom des aéroports directement au sein du diagramme.



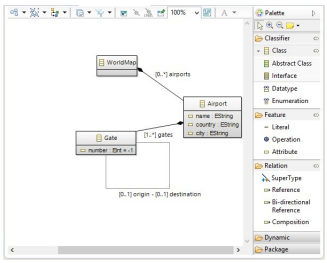
6 Conclusion et perspectives

Et voilà ! Au terme de cet article, vous saurez spécifier un premier éditeur de diagrammes avec Sirius ! Comme vous avez pu le constater, la mise en place, une fois les mécanismes compris, est rapide et ne nécessite aucune ligne de code. Néanmoins, il est possible d'utiliser ses propres « services » développés directement en Java au sein de l'éditeur.

Pour aller plus loin, je vous invite à lire la documentation complète. La vue « Propriétés » permet d'éditer les éléments du diagramme grâce à la vue par défaut d'EMF. Afin d'avoir un rendu plus agréable et plus ergonomique, l'utilisation d'EEF, « Extended Editing Framework », permet de créer une vue « Propriétés » avec des onglets de la même forme que celles dont on dispose dans la spécification de l'éditeur Sirius. Là encore la construction se fait de manière déclarative. Il est intéressant aussi de noter que Sirius a permis de refondre l'éditeur de modèles Ecore afin de lui donner une nouvelle jeunesse et sera déployé dans Eclipse Luna. Ci-dessous, deux captures d'écran du même modèle dans l'éditeur actuel et dans celui basé sur Sirius. Par exemple, on peut y créer directement des interfaces, classes abstraites, ou des relations symétriques.



Editeur de diagramme Ecore actuel



Editeur de diagramme Ecore avec Sirius

7 Liens

— Site officiel du projet : lien 17 ;
— Documentation complète Sirius lien 18 ;
— Site officie Acceleo : lien 19 ;
— Extended Editing Framework : lien 20.

*Retrouvez l'article d'**Alain Bernard** en ligne : lien 21*

Java

Les dernières news Java

Java : le projet Jigsaw se concrétise, Oracle passe à la mise en œuvre de la modularité dans Java 9

Java aura droit à son système de modules. Le projet Jigsaw prend forme après plusieurs problèmes techniques qui ont entraîné son report et sa réimplémentation.

Pour rappel, le projet Jigsaw vise essentiellement à découper la bibliothèque d'exécution de base de Java en différents modules. Cela devrait permettre à une machine virtuelle Java (JVM) de fonctionner sans le support de certains packages de base.

Après un travail préliminaire qui avait abouti à une proposition en 2008, le système de modules du projet Jigsaw allait être l'une des principales innovations de Java 7. Le projet avait été ensuite reporté à Java 8, puis finalement, Oracle avait annoncé sa sortie avec Java 9, prévu pour début 2016.

Les principales raisons de ces reports étaient des problèmes techniques liés à la compatibilité. La mise au point d'un système de modularité dans le JDK est un véritable défi, à cause des dépendances qui existent entre les packages. Par exemple, java.beans à une dépendance vers java.applet, qui dépend à son tour de java.awt.

Les difficultés rencontrées avaient entraîné, en septembre dernier, la création d'un nouveau prototype de Jigsaw permettant d'explorer une approche simplifiée pour la réalisation des objectifs de ce projet. Une décision qui n'avait pas manqué de semer à nouveau le doute sur l'effectivité de Jigsaw.

Mark Reinhold, architecte en chef du groupe de la plateforme Java chez Oracle vient d'apporter de bonnes nouvelles par rapport à l'évolution du projet Jigsaw, qui est passé en phase 2. Après une phase exploratoire de plusieurs années, Reinhold annonce qu'il est temps de passer à la vitesse supérieure.

L'équipe en charge du projet va désormais se concentrer sur la mise en œuvre de Jigsaw pour JDK 9 et Java SE 9. Reinhold a rédigé un nouveau document qui présente les objectifs et les exigences du système, en prenant en compte les enseignements tirés du dernier prototype. Le projet devra :

- rendre la plateforme Java SE et le JDK plus facilement extensible, pour cibler les petits appareils informatiques ;
- améliorer la sécurité et la maintenabilité de la plateforme Java SE et du JDK ;
- permettre d'améliorer les performances des applications ;
- faciliter le développement et la maintenance des bibliothèques et grandes applications pour les développeurs.

Reinhold reste cependant prudent, car les difficultés sont encore énormes. *« Jigsaw dans son ensemble apportera de gros changements au JDK. Il est important d'attendre que tout soit complètement finalisé avant de fusionner tout cela »*, a écrit Reinhold.

Le groupe de travail sur le projet a l'intention de procéder en quatre grandes étapes dont chacune aura droit à un JEP (JDK Enhancement Proposal). Les trois premières grandes étapes permettront de proposer une structure modulaire spécifique pour le JDK, de réorganiser le code du JDK et de procéder plus tard à la modularisation des binaires. La quatrième étape présentera le système de module proprement dit, sous une forme que les développeurs pourront utiliser pour d'autres programmes.

Mark Reinhold se veut confiant, mais jusqu'ici, il n'est pas encore certain que le projet Jigsaw sera effectif avec Java 9. Espérons que l'on n'aura pas encore droit à un nouveau report.

L'annonce de Mark Reinhold : lien 22

Le nouveau document de spécification des objectifs et exigences : lien 23

Et vous ?

Qu'en pensez-vous ? Êtes-vous impatient de voir Jigsaw dans Java ? Pourquoi ?

Craignez-vous un nouveau report ?

Commentez la news de ***Hinault Romaric*** *en ligne : lien 24*

Les derniers tutoriels et articles

Tutoriel pour développer un batch Java avec EasyBatch en moins de 5 minutes

Le but de cet article est d'introduire le framework Easy Batch (lien 25) et de montrer comment il peut faciliter le développement de batchs en Java.

1 Introduction

Dans cet article, je vais essayer de présenter comment le framework Easy Batch peut vous aider à implémenter rapidement vos batchs en Java. Tout d'abord, je vais commencer par donner un aperçu général sur Easy Batch, puis nous allons voir une étude de cas qui mettra en pratique le développement d'un batch type en utilisant Easy Batch.

Easy Batch est un framework qui permet de faciliter le développement d'application de type batch en Java. L'idée du framework est de décharger le développeur des tâches fastidieuses (lecture des données, leur filtrage, leur parsing, leur mapping à des objets du domaine et leur validation) et de le laisser se concentrer sur la logique métier de son application. La figure 1 représente les motivations derrière le framework Easy Batch :

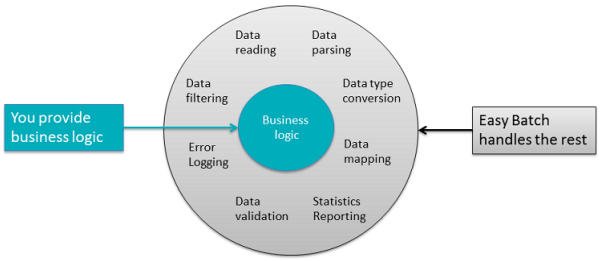


Figure 1 : Motivations du framework Easy Batch

Dans le vocabulaire Easy Batch, un record représente un item de la source de données. Un record peut être une ligne dans un fichier plat, une balise dans un fichier xml, un tuple d'une table de base de données, etc. Easy Batch traite une source de données en plusieurs étapes comme illustré dans la figure 2 :

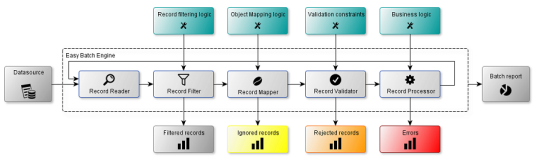


Figure 2 : Architecture d'Easy Batch

1. Lecture des données record par record : la lecture des données est faite par le *RecordReader* ;
2. Filtrage des données : il est souvent nécessaire de filtrer des données en entrée selon un ensemble de critères. Par exemple, filtrer une ligne de commentaire dans un fichier plat. Le filtrage des données est effectué par le composant *RecordFilter* ;
3. Projection dans le monde objet : généralement, on est amené à se transposer dans le monde objet en manipulant une vue objet d'un record de la source de données. Il existe plusieurs projets Java qui se basent sur cette approche (Object-Relational Mapping, Object-XML mapping, Object-JSON mapping, etc). Avec Easy Batch, il est possible de mapper un record de la source de données vers un objet du domaine en utilisant le *RecordMapper* ;
4. Validation des données : la validation des données est une étape importante dans toute application d'entreprise. Easy Batch permet de valider les données avec le composant *RecordValidator* avant de les traiter ;
5. Traitement des données : enfin, le traitement proprement dit des données se fait dans le *RecordProcessor* ;
6. À la fin d'exécution, Easy Batch fournit un rapport et un fichier de logs permettant de voir les lignes traitées avec succès, les lignes traitées avec erreurs, le temps d'exécution, etc.

À part la lecture de données, toutes les étapes sont optionnelles. Easy Batch fournit des implémentations par défaut de chaque composant. Comme illustré dans la figure 2, ces composants sont totalement configurables pour s'adapter aux besoins.

Généralement, avec Easy Batch il suffit de fournir la logique métier de l'application en implémentant l'interface *RecordProcessor* et de déléguer le reste des tâches (lecture, filtrage, mapping et validation des données) à la charge du framework.

2 Cas d'utilisation

Considérons qu'on veut traiter un fichier CSV en entrée contenant des données de produits afin de les charger dans une base de données. Le fichier CSV est le suivant :

```
id,name,description,price,published,lastUpdate
1,product1,description1,2500,true,2014-01-01
2,product2,description2,2400,true,2014-01-01
```

Imaginons que nous avons une classe *Product* qui représente la vue objet d'une ligne CSV :

```
public class Product {

    private int id;
    private String name;
    private String description;
    private double price;
    private boolean published;
    private Date lastUpdate;
    //getters et setters omis

}
```

Le but est de créer une instance du type *Product* pour chaque ligne CSV, puis de stocker le produit dans la base de données (avec un *PersistenceManager* JPA par exemple). Avant de conserver les données en base, il faudra les valider afin de vérifier que :

1. L'identifiant du produit n'est pas nul ;
2. Le nom du produit n'est pas vide ;
3. Le prix du produit n'est pas négatif ;
4. La date de mise à jour du produit est dans le passé.

Dans cet article, pour rester simple, nous allons plutôt afficher un message dans la console au lieu de stocker les données en base. Enfin, nous voulons avoir à la fin d'exécution du batch un rapport précisant les lignes traitées avec succès et avec erreurs. Voici donc ce que nous avons à faire afin d'implémenter tous ces besoins :

1. Lire le fichier CSV ligne par ligne ;
2. Mapper chaque ligne à une instance du type Product ;
3. Valider les données de produit ;
4. Stocker le produit en base de données ;
5. Générer un rapport d'exécution.

Avant de continuer, notons que dans tout ce qu'il y a à faire ci-dessus, seul le stockage des données en base représente la logique métier de notre application. Tout le reste peut être délégué à un outil (Easy Batch en l'occurrence) en fournissant quelques métadonnées de configuration (logique de mapping objet, contraintes de validation des données, etc.). Voyons donc comment utiliser Easy Batch pour traiter ce genre de besoin. Allez, top chrono !

2.1 Lecture du fichier

Notre besoin est de lire les données d'un fichier plat. Avec Easy Batch, il suffit d'utiliser le *FlatFileRecordReader* conçu spécialement pour ça :

```
EasyBatchEngine easyBatchEngine = new EasyBatchEngineBuilder()
    .registerRecordReader(new FlatFileRecordReader(new File("products.csv")))
    .build();
```

Par défaut, le *FlatFileRecordReader* utilise la virgule comme séparateur, ce qui nous convient dans cet exemple. Le séparateur ainsi que d'autres paramètres sont configurables.

2.2 Mapping des données à des objets de type Product

Pour mapper chaque ligne CSV à une instance de type *Product*, il suffit d'utiliser le *DelimitedRecordMapper* :

```
EasyBatchEngine easyBatchEngine = new EasyBatchEngineBuilder()
    .registerRecordReader(new FlatFileRecordReader(new File("products.csv")))
    .registerRecordMapper(new DelimitedRecordMapper(Product.class))
    .build();
```

Par défaut, Easy Batch va se baser sur la première ligne du fichier pour mapper chaque champ CSV à la propriété de l'objet *Product* ayant le même nom. Noter qu'Easy Batch fait aussi la conversion des données en fonction du type du champ cible. La stratégie de mapping est totalement configurable et peut être adaptée au besoin.

2.3 Validation des données

Pour valider les données, Easy Batch fournit une implémentation de l'interface *RecordValidator* qui utilise l'API Bean Validation (JSR303) afin de valider les données. Cette API est très élégante dans le sens où elle nous permet de **déclarer** les contraintes de validation et non de les implémenter. Pour valider les données de produit comme demandé, il suffit donc d'annoter le type *Product* avec les annotations de l'API Bean Validation :

```
public class Product {

    @NotNull
    private long id;

    @NotEmpty
    private String name;

    private String description;
```

```
11      @Min(0)
12      private double price;
13
14      private boolean published;
15
16      @Past
17      private Date lastUpdate;
18
19      // getters et setters omis
20
21  }
```

Puis d'enregistrer le *BeanValidationRecordValidator* dans Easy Batch :

```
1  EasyBatchEngine easyBatchEngine = new
       EasyBatchEngineBuilder()
2      .registerRecordReader(new
           FlatFileRecordReader(new File("
           products.csv")))
3      .registerRecordMapper(new
           DelimitedRecordMapper(Product.
           class))
4      .registerRecordValidator(new
           BeanValidationRecordValidator<
           Product>())
5      .build();
```

2.4 Implémentation de la logique métier

À cette étape, il faudra quand même écrire un peu de code :-) (ce qu'easy batch ne peut pas deviner). La logique métier de notre application veut que l'on stocke les données en base de données, mais comme je l'ai dit un peu plus haut, pour rester simple, nous allons juste afficher dans la console les données du produit traité. Pour cela, il suffit d'implémenter l'interface *RecordProcessor* ou étendre la classe utilitaire *AbstractRecordProcessor* fournie par Easy Batch :

```
1  public class ProductProcessor extends
       AbstractRecordProcessor<Product> {
2      public void processRecord(final
           Product product) throws
           Exception {
3          System.out.println("product = "
               + product);
4      }
5  }
```

Puis d'enregistrer notre *ProductProcessor* dans Easy Batch :

```
1  EasyBatchEngine easyBatchEngine = new
       EasyBatchEngineBuilder()
2      .registerRecordReader(new
           FlatFileRecordReader(new File("
           products.csv")))
3      .registerRecordMapper(new
           DelimitedRecordMapper(Product.
           class))
4      .registerRecordValidator(new
           BeanValidationRecordValidator<
           Product>())
5      .registerRecordProcessor(new
           ProductProcessor())
6      .build();
```

2.5 Générer un rapport d'exécution

Pour générer un rapport d'exécution, nous n'avons rien à faire, Easy Batch se charge du reporting. Nous pouvons récupérer le résultat du batch à la fin de l'exécution dans une instance du type *EasyBatchReport* :

```
1  EasyBatchReport easyBatchReport =
       easyBatchEngine.call();
```

Le rapport d'exécution contient plusieurs informations, entre autres :

1. La source des données traitées ;
2. Les lignes ignorées, les lignes traitées avec erreurs (avec la cause de l'erreur) et les lignes traitées avec succès ;
3. Le temps total d'exécution et le temps d'exécution par ligne ;
4. Etc.

Voilà, nous avons fini notre implémentation avec Easy Batch. Comme vous l'avez vu, nous n'avons implémenté que la logique métier de notre application, Easy Batch nous a déchargé de tout le reste, à savoir :

1. Lecture des données ligne par ligne ;
2. Parsing des données CSV ;
3. Mapping des données à des instances de notre objet de domaine *Product*
4. Validation des données ;
5. Génération d'un rapport d'exécution.

3 Conclusion

Dans cet article, nous avons vu comment le framework Easy Batch peut faciliter le développement de batchs en Java, en déchargeant le développeur des tâches fastidieuses et en le laissant se concentrer sur la logique métier de son application.

Finalement, à part la logique métier de notre application, nous n'avons écrit que cinq lignes de code pour configurer Easy Batch. En réalité, sans Easy Batch, nous aurions écrit une centaine de lignes de code (lien 26) pour implémenter les besoins ci-dessus. Le gain est donc considérable, non seulement en termes de temps de développement, mais aussi de maintenabilité, de possibilité de réutilisation et de clarté du code.

Le code source de cet article peut être téléchargé ici : lien 27. Pour exécuter le tutoriel, il suffit de décompresser l'archive puis de lancer la commande Maven suivante :

```
1  mvn package exec:java -PrunProductSample
```

4 Annexe - Comparaison avec Spring Batch et la JSR 352

Spring Batch est aujourd'hui la référence pour le traitement des données par lots en Java. La popularité de ce framework a conduit à la standardisation de ses concepts dans la JSR 352 (Batch Applications for the Java Platform) de l'API Java EE 7. Spring Batch est très riche en fonctionnalités mais a une courbe d'apprentissage importante et n'est pas toujours facile à mettre en place. Il arrive même que sa mise en place soit plus compliquée que la résolution du problème !

Easy Batch, de son côté, se veut une alternative plus légère qui n'a pas vocation à concurrencer Spring Batch, mais qui peut être utilisée lorsque Spring Batch s'avère une solution lourde et coûteuse pour le problème à résoudre. Dans cette annexe, je vais essayer de comparer objectivement les deux frameworks en me basant sur les critères suivants :

	Easy Batch	**Spring Batch / Java EE 7 JSR 352**
Implémente la JSR 352	Non	Oui
Taille de la librairie	40Ko	543Ko
Courbe d'apprentissage	Faible	Grande
Développement basé sur des Pojos	Oui	Oui
Partitionnement de données	Oui	Oui
Supervision en temps réel	Oui	Oui
Gestion des transactions	Programmatique	Déclarative
Reprise sur erreur	Non	Oui
Administration à distance	Non	Oui
Traitement par lot (chunk)	Non	Oui
Traitement en parallèle	Oui	Oui
Traitement asynchrone	Oui	Oui
Configuration de Job	Java	Java, Xml, annotation
Ordonnancement de Job	Oui	Oui

La figure 3 illustre la position d'Easy Batch entre Spring Batch et une solution " Partir de zéro " :

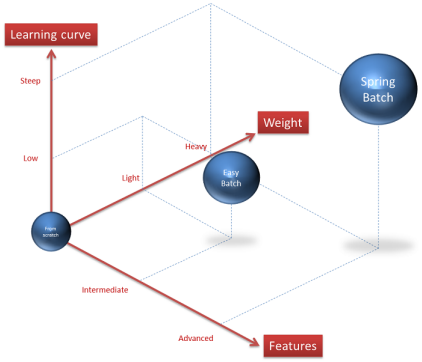


Figure 3 : Easy Batch versus Spring Batch

En termes de fonctionnalités, Spring Batch (et les implémentations de la JSR 352) fournit plus de fonctionnalités avancées (remoting, flows, etc.).

Par contre, Easy Batch est une solution plus légère et plus facile à apprendre et à mettre en place. Noter aussi que les amateurs de la nouvelle API Batch auront besoin de monter un serveur d'application pour pouvoir traiter un fichier plat :-). Cette idée est choquante pour pas mal de gens et Easy Batch n'a pas été conçu pour ce mode de fonctionnement.

Pour conclure cette comparaison, Easy Batch est une solution intermédiaire entre Spring Batch et la solution de partir de zéro pour développer un batch en Java. Vous pouvez aussi trouver une comparaison complète entre Spring Batch et Easy Batch avec un exemple concret dans ce post : lien 28.

5 Références

1. Page GitHub du projet : lien 29 ;
2. Documentation en ligne : lien 30 ;
3. Présentation du projet : lien 31.

Retrouvez l'article de ***Mahmoud Ben Hassine*** *en ligne : lien 32*

JavaScript

Les derniers tutoriels et articles

Meteor : la plateforme Web temps réel qui accroît la productivité

Le Web a connu de nombreuses évolutions ces dernières années. HTML5, WebSocket et Node.js sont des termes que l'on rencontre désormais souvent. Les frameworks JavaScript ont plus que jamais le vent en poupe, maintenant que le langage ne se limite plus nécessairement à la partie client. Il n'est cependant pas toujours facile de savoir par où (re)commencer. Meteor est une plateforme JavaScript permettant de construire rapidement des applications Web modernes. Elle a l'avantage d'être facile à apprendre et, en plus, elle est « full-stack », ce qui veut dire que vous n'avez pas à mélanger plusieurs technologies pour obtenir une application entièrement fonctionnelle. Elle se destine aussi bien au client qu'au serveur. Orientée vers la productivité du développeur et le fonctionnement en temps réel, vous allez découvrir dans les lignes qui suivent qu'il s'agit d'une plateforme prometteuse qui possède de vrais atouts.

1 Introduction

Meteor est un projet open source qui a pour ambition de transformer notre manière de concevoir des applications Web. Un des principaux objectifs est d'apporter la réactivité au plus grand nombre, la possibilité de développer des applications temps réel rapidement sans avoir à sortir l'artillerie lourde.

La documentation officielle de Meteor décrit sept principes que s'efforcent de suivre ses créateurs :

- **Données sur la ligne.** On ne transmet pas du HTML mais des données et c'est le client qui décide quoi en faire ;
- **Un langage unique.** Les parties client et serveur de l'application sont écrites en JavaScript ;
- **Base de données omniprésente.** L'API pour accéder à la base de données est la même, que l'on soit sur le client ou le serveur ;
- **Compensation de latence.** Sur le client, on utilise le prérapatriement et la simulation de base de données pour donner l'impression d'une latence zéro ;
- **Réactivité de bout en bout.** Le temps réel est la norme. Toutes les couches, de la base de données au template, doivent proposer une interface orientée événement ;
- **Intégration à l'écosystème.** Meteor est open source et intègre les outils et frameworks existants plutôt que de les remplacer ;
- **Simplicité = Productivité.** Le meilleur moyen de donner l'impression de simplicité est de faire quelque chose qui soit réellement simple et cela passe par la création de belles API.

Techniquement, Meteor s'appuie sur Node.js pour l'exécution côté serveur, MongoDB pour la base de données, les WebSockets et un protocole maison nommé DDP (*Distributed Data Protocol*) pour les communications client/serveur.

2 Installation

L'installation de Meteor est on ne peut plus simple :

```
1  curl https://install.meteor.com | /bin/
       sh
```

Cela installe l'exécutable meteor sur le système. La création d'un projet est tout aussi aisée. Puisque nous disposons dorénavant de la commande meteor, il nous suffit de lui fournir l'argument create suivi du nom que nous souhaitons donner à notre projet :

```
1  meteor create hello
```

Une fois cette commande exécutée, un dossier hello aura été créé, contenant quelques fichiers d'exemples : hello.css, hello.html et hello.js. Pour démarrer l'application, il suffit de se placer dans le

dossier fraîchement créé et de taper la commande meteor sans argument. Une fois initialisée, celle-ci sera accessible à l'URL par défaut : http ://localhost :3000/. Pour le moment notre application ne fait rien d'exceptionnel et affiche un simple *Hello World*. Nous allons tout de même jeter un œil au code qui a été généré.

3 Premier contact

Nous allons ouvrir les fichiers fraîchement créés afin de voir ce qu'ils contiennent. Le fichier CSS est vide, commençons donc par analyser le fichier HTML :

```
<head>
    <title>hello</title>
</head>
<body>
    {{> hello}}
</body>
<template name="hello">
    <h1>Hello World!</h1>
    {{greeting}}
    <input type="button" value="Click" />
</template>
```

Nous pouvons déjà voir qu'il ne s'agit pas seulement de HTML mais que certains éléments semblent appartenir à la syntaxe d'un moteur de template. Et c'est tout à fait juste puisque Meteor utilise son propre moteur de template, nommé **Spacebars**. Nous pouvons donc voir une expression greeting (notez les accolades doubles et) et la définition d'un template à l'aide de la balise <template> comprenant un attribut name devant contenir l'identifiant du template qui sera référencé pour affichage (en l'entourant par > et).

Intéressons-nous maintenant à l'endroit où l'expression greeting est définie et ouvrons le fichier JavaScript :

```
if (Meteor.isClient) {
    Template.hello.greeting = function () {
        return "Welcome to hello.";
    };

    Template.hello.events({
        'click input' : function () {
            // template data, if any, is available in 'this'
            if (typeof console !== 'undefined')
                console.log("You pressed the button");
        }
    });
}

if (Meteor.isServer) {
    Meteor.startup(function () {
        // code to run on server at startup
    });
}
```

La première chose qui devrait vous frapper est l'utilisation de Meteor.isClient et Meteor.isServer qui indiquent que ce code est destiné à la fois au serveur et au client. Je vois déjà les plus paranoïaques (ou consciencieux ?) d'entre vous s'étrangler avec leur café à la vue de ce qui semble être une aberration. Ne vous inquiétez pas, il ne s'agit que d'un exemple et en réalité vous ne serez pas obligés d'exposer votre code serveur au public. C'est même plutôt déconseillé. Il reste toutefois possible de partager du code entre le client et le serveur. C'est utile pour certaines bibliothèques ou éléments de configuration. Je reviendrai ultérieurement sur la manière de structurer votre application afin de séparer votre code serveur du code client.

Si on observe le code destiné au client, on voit tout d'abord la définition de l'expression greeting que nous avions vue plus haut. C'est une simple fonction qui renvoie du texte. Vous remarquerez que la fonction a été ajoutée à l'objet Template.hello. Template est un objet mis à disposition par Meteor. Il contient l'ensemble des instances de template dont celui nommé hello qui a été déclaré dans le fichier HTML. Enfin, un gestionnaire d'événement a été ajouté au template hello à l'aide de la méthode events(). Celle-ci prend un objet littéral dont les clefs sont des chaînes de caractères définissant l'événement. Ici, un clic sur l'élément input du template hello entraîne l'apparition d'un message dans la console.

La dernière partie du fichier fait appel à Meteor.startup() pour exécuter une fonction au démarrage du serveur. Actuellement, elle ne fait rien et n'est là qu'à titre d'exemple.

Nous avons fait le tour du code d'exemple. Mais nous n'avons vu ni réactivité ni quoi que ce soit qui ait un rapport avec du temps réel. Avant cela, nous allons aborder quelques points fondamentaux. Vous aurez remarqué jusqu'ici que l'API de Meteor est partagée par le code serveur et le code client. Il faut savoir qu'il existe cependant des différences dans le comportement et la disponibilité de certaines méthodes ou propriétés en fonction de l'endroit où elles sont appelées. La documentation détaille, pour chaque élément de la plateforme, sa disponibilité (client, serveur ou n'importe où) et son fonctionnement dans les deux cas.

4 Structurer son application

Une application Meteor réunit à la fois du code JavaScript exécuté dans un conteneur Node.js, du code client destiné au navigateur, des fragments de HTML, du style CSS et diverses ressources statiques. Meteor laisse à chacun le choix de l'organisation de son application, mais il y a un certain nombre de choses à savoir pour ne corrompre ni le fonctionnement, ni la sécurité de son application. Meteor s'appuie notamment sur le nom de vos répertoires et de vos fichiers pour déterminer à qui est destiné leur contenu et dans quel ordre ils seront chargés. Il est donc crucial de savoir comment Meteor traite votre arborescence. Globalement, ce sont trois critères qui vont déterminer le nom et l'emplacement que vous donnerez à un fichier au sein d'une application Meteor : le rôle qu'il joue dans votre application, sa visibilité et l'ordre dans lequel il doit être chargé.

4.1 Visibilité

L'endroit où vous déciderez de placer vos fichiers va dépendre de la visibilité que vous souhaitez leur donner. Il y a donc trois possibilités : visibles par le serveur, visibles par le client ou les deux. La partie la plus sensible de votre code sera probablement accessible uniquement par le serveur, tandis que ce qui concerne l'interface sera plutôt destiné au client. Enfin, vous voudrez peut-être partager certaines bibliothèques à la fois avec le client et le serveur.

Les fichiers contenus dans les répertoires server et private sont invisibles pour le client et sont donc uniquement accessibles au serveur. À l'inverse, ce qui est contenu dans le répertoire client et public ne sera pas pris en compte sur le serveur.

Tous les fichiers se trouvant ailleurs sont visibles dans les deux cas, à l'exception de ce qui est contenu dans le répertoire tests, qui est un répertoire spécial dont le contenu ne sera pas déployé avec l'application.

4.2 Rôle

Chaque fichier de votre application joue un rôle précis. Nous pouvons identifier les éléments suivants : le code JavaScript, les fragments de HTML, les mises en forme CSS et des ressources statiques (images, fichiers divers, etc.).

4.2.a Code JavaScript

Tout code JavaScript est exécuté à condition qu'il ne se trouve pas dans un répertoire private, public ou tests.

4.2.b HTML

La manière dont le HTML est traité est assez particulière. Meteor scanne l'ensemble de vos fichiers HTML et en extrait les éléments <head>, <body> et <template>. Les contenus des éléments <head> et <body> sont fusionnés et placés respectivement dans des éléments <head> et <body> uniques qui seront transmis au client au chargement initial de la page. Les éléments <template> sont, quant à eux, convertis en fonctions JavaScript et rendus accessibles via l'objet Template. Cela est valable pour les fichiers ne se trouvant pas dans un répertoire server, private ou public.

4.2.c CSS

Tous les fichiers CSS sont fusionnés et envoyés au client comme une seule feuille de style, à l'exception de ceux se trouvant dans les répertoires server, public et private.

4.2.d Ressources statiques

Les répertoires public et private sont dédiés aux ressources statiques. Le contenu du répertoire public est rendu accessible au client comme tout répertoire public d'une application Web. Vous pourrez y placer vos images, favicon.ico, robots.txt, etc. Les fichiers du répertoire private sont rendus accessibles au code serveur via l'API Assets qui dispose de méthodes pour en récupérer le contenu.

4.3 Priorité

Meteor encourage autant que possible les développeurs à écrire leur code de manière à ce qu'il soit le moins possible sensible à l'ordre de chargement des fichiers. Il est pourtant rare de ne pas avoir à s'en soucier. Voici donc les règles à connaître pour éviter quelques maux de tête :

- les fichiers se trouvant dans des sous-répertoires sont chargés avant ceux qui se trouvent dans des répertoires parents ;
- dans un même répertoire, les fichiers sont chargés par ordre alphabétique ;
- une fois ces deux premières règles appliquées, les fichiers se trouvant dans des répertoires nommés lib sont placés avant tout le reste ;
- les fichiers correspondant au pattern main.* sont placés à la fin.

Ces règles de chargement peuvent ne pas vous convenir. Il est donc à noter que si vous tenez particulièrement à définir l'ordre de chargement exact de vos fichiers, il vous faudra constituer un package. En effet, le fichier de configuration d'un package permet non seulement de définir les dépendances avec d'autres packages, mais aussi l'ordre dans lequel doivent être chargés les fichiers qu'il contient. Pour en apprendre plus sur la création d'un package, je vous invite à consulter la documentation de Meteor.

5 Définir son modèle de données

Les données sont au centre de toute application. Ce qui différencie toutefois une application Meteor d'une application plus classique est la manière dont est abordé le modèle de données. Ce que j'entends par modèle de données dans ce cas précis est, non seulement la définition et la représentation des données, mais aussi la manière dont l'application va les traiter et les acheminer au client.

Dans une application classique vous aurez probablement tendance à vous concentrer en premier sur la définition de vos données et sur la manière de les stocker. Une fois que votre schéma sera suffisamment mûr, vous vous lancerez dans le squelette de votre application en suivant l'habituel pattern MVC (Modèle - Vue - Contrôleur). À chaque fois que vous implémenterez une nouvelle fonctionnalité, vous vous demanderez quelle partie de vos données est concernée, qui pourra y avoir accès et dans quelles circonstances. Ces questions, qui vous viennent souvent dans un second temps, Meteor vous amène à y réfléchir dès le départ.

5.1 Une base de données omniprésente

Contrairement à ce qui se fait couramment dans nos applications Web, le serveur Meteor ne se contente pas de transmettre du HTML au client. Au lieu de ça, Meteor envoie des données « brutes » et c'est au client de les traiter. Nous allons éclaircir ce point.

La plateforme Meteor s'efforce de mettre à disposition une API uniforme côté client et serveur. C'est tout particulièrement vrai en ce qui concerne l'accès à la base de données. Ce qui veut dire qu'il est tout autant possible d'accéder à la base de données depuis le client que depuis le serveur. Je ne doute pas que cette révélation amène un certain nombre de questions. Comment les requêtes sont-elles transmises à la base de données et quid de la sécurité ?

La première chose qu'il faut savoir est que Meteor utilise une bibliothèque nommée **minimongo** pour simuler une base de données MongoDB côté client. Concrètement, c'est une copie de la véritable base de données qui est envoyée à chaque client. Cette copie n'est généralement pas la version intégrale de la base de données d'origine, mais contient les données que l'on souhaite rendre accessibles au client. Celui-ci peut donc effectuer des requêtes sur sa propre copie de la base de données en utilisant la même API que sur le serveur.

Pour ceux qui ne seraient pas familiarisés avec MongoDB, il s'agit d'une base de données orientée documents. Un document équivaut à un enregistrement composé de divers champs. La principale différence avec une base de données relationnelle est que ces champs ne sont pas figés, c'est pourquoi on parle de base de données *schema-less* (sans schéma donc). Un ensemble de documents est appelé « collection » et c'est l'équivalent d'une table dans une base de données relationnelle.

5.2 Les collections Meteor et l'API partagée

La déclaration d'une collection intitulée messages se fait de la manière suivante :

```
1  Messages = new Meteor.Collection("
       messages");
```

En général, on placera la déclaration d'une collection dans un fichier interprété sur le serveur et le client car c'est un élément commun. Il est ensuite possible d'effectuer des requêtes sur la collection via des méthodes comme find(), findOne(), insert(), update() ou encore remove(). L'API des collections est partagée entre le client et le serveur, il est donc possible d'effectuer des requêtes de manière identique des deux côtés.

La méthode find() permet de récupérer les documents d'une collection qui correspondent au sélecteur MongoDB qui, lui, est passé en paramètre. Pour rechercher des documents selon un champ userId, on procéderait de la manière suivante :

```
1  var myMessages = Messages.find({userId:
       Session.get('myUserId')}).fetch();
```

Vous remarquerez l'usage de la méthode fetch() sur le résultat. Celle-ci permet de récupérer les résultats d'un curseur sous la forme d'un tableau car la méthode find() renvoie en fait un curseur. De la même manière, vous pouvez utiliser la méthode count() sur un curseur pour connaître le nombre de résultats ou la méthode forEach() pour itérer sur les documents retournés :

```
1  var myMessages = Messages.find({userId:
       Session.get('myUserId')});
2  console.log(myMessages.count() + "
       messages found !");
3  myMessages.forEach(function (message) {
4     console.log("Subject of message : " +
            message.subject);
5  });
```

En plus d'un sélecteur, la méthode find() peut prendre un certain nombre d'options dont les plus courantes sont le tri, l'offset, la limite et les champs à retourner. L'exemple suivant montre comment on récupérerait une liste de messages triés par date de création descendante, appartenant à un utilisateur connu, en limitant le nombre de résultats à 10.

```
1  var myMessages = Messages.find({userId:
       Session.get('myUserId')}, {sort: {
       crea
2  ted_at: -1}, limit: 10});
```

De manière similaire à la méthode find(), la méthode findOne() permet de récupérer un seul résultat, le premier qui correspond au sélecteur. La principale différence est qu'ici, ce n'est pas un curseur qui est renvoyé, mais directement le document qui a été trouvé :

```
var myMessage = Messages.findOne({userId : Session.get('myUserId')});
```

Pour insérer un document dans une collection, vous utilisez la méthode insert(), qui prend en paramètre l'objet à insérer. Cette fonction renvoie l'identifiant du document qui a été inséré.

```
var messageId = Messages.insert({subject : "un titre", content: "du texte"});
```

Un des avantages d'une API unique est probablement que vous n'interrompez pas la progression de votre travail quand vous développez la partie client, puisque vous n'aurez pas nécessairement à faire appel à des procédures distantes (ce qui reste toutefois possible). Vous restez concentrés sur la logique que vous implémentez et non sur les moyens d'y parvenir. Vos requêtes étant effectuées sur une copie locale de la base de données, vous aurez de plus l'impression d'une latence zéro car les changements sont visibles immédiatement et propagés « en arrière-plan » sur le serveur. Sachez toutefois que le serveur peut tout à fait rejeter une modification et remettre votre copie locale à son état initial. Cela peut provoquer de drôles d'effets visuels si vous n'y êtes pas préparés. Toutefois, vous vous en doutez, il y a quelques précautions à prendre en matière de sécurité et de volume de données à transmettre à chaque client.

5.3 Publications et souscriptions

Il n'est pas souhaitable que l'intégralité de la base de données soit copiée chez chaque client. C'est la raison d'être des publications. Une publication est déclarée côté serveur et permet d'indiquer à Meteor quel sous-ensemble de vos données vous souhaitez envoyer à chaque client. Pour cela on utilise la méthode Meteor.publish() dont le premier argument est un nom arbitraire à donner à votre publication (qui sera utilisé par les souscriptions côté client) et le second une fonction qui renvoie un curseur (ou un tableau de curseurs) pour les documents pouvant être publiés.

```
Meteor.publish("myMessages", function () {
   return Messages.find({userId: this.userId});
});
```

L'expression this.userId contient l'identifiant de l'utilisateur connecté (ou null) quand utilisé au sein d'une publication. Une fonction de publication est évaluée à chaque fois qu'un client souscrit à cette publication. Cette souscription se déclare ainsi, côté client :

```
Meteor.subscribe("myMessages");
```

Le nom indiqué à la souscription doit correspondre à la publication souhaitée. Il est tout à fait possible de déclarer plusieurs publications portant sur la même collection. Un client pourra alors souscrire à l'une ou l'autre, ou même plusieurs en même temps (dans ce dernier cas les documents seront fusionnés).

Afin que vos publications soient prises en compte, il faut prendre soin de supprimer le package **autopublish**. Ce package est installé par défaut. Son rôle est de publier automatiquement toutes les données à tous les clients. Cela peut être utile dans certains cas, durant la phase de développement. Il faut cependant s'assurer qu'il soit supprimé avant la mise en production de votre application (et bien avant si vous voulez tester vos publications) en utilisant la commande meteor remove :

```
meteor remove autopublish
```

5.4 Restrictions d'écriture

Par défaut, un client peut mettre à jour la base de données sans restriction. C'est ce que rend possible le package insecure. Tout comme le package autopublish, il ne sert qu'à faciliter certaines opérations de débogage et de test pendant le développement, mais est à bannir en production.

```
meteor remove insecure
```

Une fois ce package supprimé, plus aucun client ne pourra écrire dans la base de données. Il faudra donner les droits d'insertion, de mise à jour et de suppression explicitement. Pour ce faire, Meteor met à disposition les méthodes allow() et deny() sur les objets de collection.

Le code ci-dessous montre comment autoriser un utilisateur à insérer, modifier et supprimer ses propres documents :

```
Messages.allow({
   insert: function (userId, doc) {
      return (userId && doc.userId === userId);
   },
   update: function (userId, doc, fields, modifier) {
      return doc.userId === userId;
   },
   remove: function (userId, doc) {
      return doc.userId === userId;
   }
});
```

Comme vous pouvez le voir, la méthode allow() accepte des callbacks pour chaque type d'opération d'écriture dans la base de données. Ceux-ci renvoient un booléen indiquant si oui ou non l'opération est autorisée. Les callbacks d'insertion et de suppression reçoivent en paramètre l'identifiant de l'utilisateur qui souhaite écrire dans la base de données et le document concerné. Le callback de mise à jour, lui, reçoit quatre paramètres. L'identifiant de l'utilisateur

passe en premier. En second, il s'agit du document à l'état avant modification. En troisième se trouvera un tableau contenant le nom des champs que le client souhaite modifier. Enfin, le dernier paramètre est le modificateur MongoDb qui a été utilisé pour effectuer la mise à jour, par exemple $set : subject : « Nouveau sujet ».

La méthode deny() fonctionne exactement de la même manière sauf que les règles définies avec deny() ont plus de poids que celles définies avec allow(). Plus exactement, en utilisant deny() vous pourrez interdire une opération même si elle a été autorisée avec allow().

6 Introduction à la réactivité

Ces dernières années, beaucoup de frameworks JavaScript ont vu le jour, chacun apportant son lot de fonctionnalités et sa vision sur la manière de structurer une application JavaScript. Cependant, l'apport qui intéresse probablement la plupart des développeurs que nous sommes, est le rafraîchissement automatique des interfaces utilisateur. Fini le temps où il fallait manuellement répercuter chaque changement d'état sur tous les éléments de votre interface. Le JavaScript moderne est capable de traquer les changements qui opèrent dans vos données et de mettre à jour automatiquement les parties de l'interface concernées. AngularJS, Knockout et React sont un exemple de bibliothèques connues pour avoir implémenté une telle fonctionnalité, que l'on trouve sous différentes appellations. C'est également le cas de Meteor qui a appelé ce concept la « réactivité ».

En réalité, dans Meteor, la notion de réactivité n'est pas restreinte à l'interface, mais j'ai choisi cette image car elle est facile à comprendre. La réactivité est la capacité d'un système à mettre à jour automatiquement ses objets quand les données dont il dépend changent. Meteor se veut être réactive de bout en bout. Cela ne veut pas pour autant dire que tout fonctionne comme par magie. Deux éléments fondamentaux entrent dans la composition d'un code réactif : le **traitement réactif** et la **source de données réactive**.

6.1 Traitement réactif

Pour qu'un code soit exécuté à chaque fois que ses dépendances changent, vous devrez le placer dans un traitement réactif. Un traitement réactif est un contexte qui contient votre code et traque ses dépendances. Lorsqu'il est notifié d'un changement, il exécute à nouveau le code dont il a la charge.

Meteor exécute le code présent dans les templates dans un traitement réactif. C'est pourquoi il est aisé de créer une interface réactive. Mais il est possible d'exécuter du code arbitraire de manière réactive côté client, à l'aide de la fonction Deps.autorun() :

```
1 Deps.autorun(function () {
2     console.log("Currently viewing
          message #" + Session.get("
          currentMessageId"));
3 });
```

Le code ci-dessus enregistre une fonction dans un traitement réactif. Cette fonction sera réexécutée à chaque fois que la valeur de Session.get(« currentMessageId ») change. Cela fonctionne car l'objet Session est une source de données réactive.

6.2 Source de données réactive

Un traitement réactif n'est pas grand-chose sans source de données réactive. C'est cette dernière qui est responsable d'informer le traitement réactif d'un quelconque changement. Nous n'allons pas entrer dans le détail de la création d'une source de données réactive, mais sachez toutefois que le package Deps, et plus particulièrement l'objet Deps.Dependency, peut vous y aider. Meteor possède déjà un certain nombre de sources de données réactives comme les variables Session, les requêtes sur des collections, Meteor.user(), Meteor.userId(), etc.

À propos de l'objet Session, je souhaite y apporter une petite précision. Contrairement à ce que laisse croire son nom, il ne s'agit pas réellement d'un objet de session comme vous pourriez le penser. Voyez cet objet comme un moyen de stocker de manière globale des ensembles de clefs-valeurs. Le principal intérêt de cet objet est qu'il est réactif et il peut vous être utile dans vos templates notamment. En appelant Session.get('something') depuis un template, le template sera rafraîchi à chaque fois que vous appellerez Session.set('something', x).

7 Préparer ses templates

Les templates sont les éléments qui vont vous permettre de composer l'interface de votre application. Un template contient classiquement du code HTML. Pour chaque vue ou élément récurrent dans votre interface, vous voudrez probablement créer un template. Pour créer un template, il suffit de créer un fichier HTML et d'y placer une balise <template> avec un attribut name :

```
<template name="hello">
   <div class="greeting">Hello {{name}}!
       </div>
</template>
```

Le contenu des fichiers HTML est automatiquement analysé et les templates rencontrés sont placés dans l'objet global Template sous le nom avec lequel ils ont été déclarés. Le template ci-dessus est ainsi accessible en appelant Template.hello. Ce composant possède des méthodes events() et helpers() pour y ajouter des événements ou des helpers. Il donne également accès aux callbacks rendered(), created() et destroyed() appelés au différentes étapes du cycle de vie d'une instance de template.

Intéressons-nous maintenant à la manière d'utiliser des templates directement dans notre code HTML, en profitant par ailleurs de la réactivité. Pour cela, nous aurons besoin d'un fichier HTML qui contienne une balise <body> afin d'y appeler notre template :

```
<body>
   {{> hello}}
</body>
```

Cela affiche notre template hello au chargement de l'application. Pour cela nous avons utilisé la syntaxe > monTemplate qui permet d'afficher le contenu d'un template. Cette syntaxe est un élément du moteur de template utilisé par Meteor, **Spacebars**, qui est un dérivé de **Handlebars**. Le moteur de template met à disposition un certain nombre d'expressions qui vous permettent de rendre vos templates dynamiques. Ces expressions sont facilement identifiables car elles sont entourées de doubles accolades et . La plupart de ces expressions sont des **helpers**.

7.1 Helpers

Nous avons vu que certains templates contenaient des expressions de la forme hello. Il peut s'agir de deux choses. Soit le template est exécuté dans un contexte où la variable hello existe et à ce moment-là elle est affichée. Cela peut être le cas si vous étiez en train d'itérer sur une collection de documents et que le document actuel (qui sera this) contient une propriété hello. Soit il existe un helper de ce nom. Sachez d'ailleurs que les helpers ont priorité si une propriété de même nom existe dans le contexte dans lequel vous vous trouvez.

La déclaration d'un helper se fait dans un fichier JavaScript au niveau du client :

```
Template.hello.name = function () {
   return "Will";
};
```

Nous venons de déclarer un helper sur le template hello qui porte le nom name et qui retourne du texte. Ainsi, le template hello affichera le contenu suivant :

```
<div class="greeting">Hello Will!</div>
```

Apportons maintenant de la réactivité à tout ça. Admettons que vous stockiez le nom de l'utilisateur actuel dans l'objet Session. C'est une source de données réactive, rappelez-vous. Nous pouvons donc faire en sorte que le template hello se rafraîchisse à chaque fois que cette donnée change :

```
Template.hello.name = function () {
   return Session.get("username");
};
```

Puisqu'il est peu courant de changer de nom, nous allons étudier un autre cas plus probable qui est l'affichage d'une liste de messages.

```
Template.hello.messages = function () {
   return Messages.find({userId: Meteor.
       userId()});
};
```

Nous profitons ici de la réactivité apportée par le curseur retourné par la méthode find(). Ce curseur étant aussi une source de données réactive, nous sommes sûrs que notre template sera réévalué à chaque fois que les résultats de cette requête changent. Modifions maintenant notre template pour utiliser notre nouveau helper :

```
<template name="hello">
   <ul>
   {{#each messages}}
      <li{{subject}}</li>
   {{/each}}
   </ul>
</template>
```

Vous remarquerez l'utilisation d'un élément nouveau, #each qui est un helper de bloc. C'est un helper particulier qui vous permet d'itérer sur une liste d'éléments. Il en existe d'autres comme #if, et il est possible d'implémenter les vôtres. Ici, nous faisons usage de #each pour parcourir les éléments renvoyés par le helper messages. Dans notre cas ce sont des documents qui possèdent une propriété subject. Vous pouvez donc constater qu'il est simple de créer des interfaces réactives en un temps limité. Il nous reste cependant à voir comment les rendre interactives.

7.2 Événements

Il est facile d'ajouter des gestionnaires d'événement à un template. Pour cela, il suffit de passer la définition de vos événements à la fonction events() que vous appellerez sur votre template de la manière suivante :

```
Template.hello.events({
   'click': function (event) {
      console.log("J'ai cliqué quelque
          part");
   },
   'click .send': function (event) {
      console.log("J'ai cliqué sur un él
          ément ayant la classe 'send'")
          ;
   },
   'keydown, click button': function (
       event) {
```

```
9         console.log("Déclenché au clavier
              ou au clic sur un bouton");
10    }
11 });
```

Le format d'un événement est simple. Vous devez spécifier le type d'événement (click, change, keydown, etc.), éventuellement associé à un sélecteur CSS pour restreindre l'événement à un élément précis. Vous pouvez séparer plusieurs événements par une virgule pour leur affecter le même gestionnaire d'événement. Le gestionnaire d'événement peut prendre en argument un objet contenant un certain nombre d'informations à propos de l'événement et des méthodes permettant d'agir sur sa propagation. Lorsque vous vous trouvez dans un gestionnaire d'événement, this contient les informations du contexte de l'élément qui l'a déclenché. Celui-ci est le contexte de données dans lequel l'élément apparaît dans le template. Cela veut dire, par exemple, que si vous itérez sur une collection de documents avec #each, et que pour chaque document vous affichez un lien, le gestionnaire d'événement prenant en charge le clic sur le lien, aura dans this le document concerné.

8 Procédures distantes

Bien qu'il soit possible d'effectuer des requêtes côté client, il arrive un moment où vous aurez besoin d'effectuer des opérations plus sensibles côté serveur. C'est notamment le cas si l'action d'un utilisateur nécessite l'accès à un ensemble de données qui n'est pas directement accessible à ce client. Pour cette raison, et beaucoup d'autres, vous voudrez créer des **méthodes Meteor** qui ne sont rien d'autre que des procédures distantes.

Pour déclarer des procédures distantes vous devez, côté serveur, faire appel à la fonction Meteor.methods() qui prend en paramètre un objet dont chaque clef est le nom de la méthode et la valeur son implémentation :

```
1 Meteor.methods({
2    add: function (a, b) {
3       return a + b;
4    },
5    substract: function (a, b) {
6       return a - b;
7    }
8 });
```

Pour invoquer une méthode côté client vous pouvez utiliser la fonction Meteor.call() ou Meteor.apply(). Elles ont la même finalité, la seule différence est que Meteor.apply() prend les paramètres à transmettre à la méthode dans un tableau et dispose de quelques options pour contrôler la manière dont la méthode est invoquée. Voici comment invoquer la méthode add qui a été définie côté serveur :

```
1 var result = Meteor.call('add', 3, 5);
```

Il vous est aussi possible de retourner une erreur dans une méthode. Pour cela, Meteor met à disposition l'objet Meteor.Error qui représente une exception qui peut être transmise au client :

```
1 Meteor.methods({
2    testError: function () {
3       throw new Meteor.Error(500, "Une
            erreur est survenue");
4    }
5 });
```

Dans une méthode Meteor, la propriété this.userId contient l'identifiant de l'utilisateur courant ou null s'il n'est pas identifié. Si vous avez implémenté vous-même une gestion de comptes utilisateur, vous pouvez définir la valeur de cette propriété dans une méthode avec this.setUserId(). Cela peut cependant être géré automatiquement avec le système de gestion de comptes mis à disposition par Meteor.

9 Utiliser les packages

Toutes les fonctionnalités de Meteor sont regroupées dans des packages. Certains font partie du cœur de Meteor et d'autres sont optionnels. Pour lister l'ensemble des packages disponibles vous pouvez tout simplement utiliser la commande meteor list. Pour filtrer sur les packages installés dans votre application, utilisez l'argument –using.

```
1 meteor list --using
```

L'ajout et la suppression d'un package se font avec les commandes meteor add <package> et meteor remove <package>. En plus des packages standards livrés avec Meteor, vous pouvez placer des packages tiers dans le répertoire packages à la racine de votre application. Ces packages tiers peuvent très bien être les vôtres, si vous écrivez votre application de manière modulaire.

Les packages vous permettent de disposer de fonctionnalités supplémentaires avec un minimum d'efforts. Cela vous permet de vous concentrer sur ce qui est réellement important. Le meilleur exemple qui pourrait montrer l'intérêt de l'usage des packages est l'intégration du système de gestion de comptes Meteor dans votre application.

9.1 Gestion de comptes utilisateur

Meteor dispose d'un système de gestion de comptes utilisateur mis à disposition à travers ses packages standards. Ces packages ne sont pas intégrés par défaut mais vous pouvez les ajouter à tout moment. Meteor a fait en sorte de séparer chaque aspect de la gestion de comptes dans un package différent afin que vous puissiez sélectionner ce dont vous avez réellement besoin. Le cœur du système se trouve donc dans le package **accounts-base**. Celui-ci est ajouté automatiquement quand vous intégrez un des gestionnaires d'authentification mis à disposition. Parmi ces gestionnaires d'authentification on trouve **accounts-password** qui est le système vous permettant de gérer l'authentification par mot de passe. Il en existe d'autres qui apportent l'authentification via des services tiers comme **accounts-facebook**, **accounts-google** et **accounts-twitter**.

Pour l'exemple, nous allons ajouter la gestion de comptes par mot de passe dans une application existante :

```
1 meteor add accounts-password
```

En faisant cela, le package accounts-base est ajouté implicitement puisqu'il s'agit d'une dépendance du package accounts-password. Le package accounts-base, qui est le cœur du système, ajoute la gestion de documents utilisateur dans la base de données. Il apporte également l'intégration de la propriété userId dans les publications et les méthodes Meteor que nous avons vues précédemment. Enfin, il met à disposition une API pour accéder aux documents utilisateur et gérer l'authentification.

Le package accounts-password apporte, lui, une API pour gérer très simplement l'authentification et la création d'utilisateurs avec mot de passe. Vous aurez donc accès à des fonctions comme Accounts.createUser(), Accounts.forgotPassword(), Accounts.verifyEmail() et d'autres pour implémenter l'enregistrement et la connexion d'utilisateurs.

Meteor va même jusqu'à proposer un package **accounts-ui** qui va mettre à disposition une interface par défaut pour les différents gestionnaires d'authentification présents dans votre application. Vous aurez alors accès à un template loginButtons que vous placerez dans votre HTML à l'endroit où vous souhaitez voir apparaître le widget apportant tous les contrôles pour la connexion ou l'enregistrement d'utilisateurs.

Pour profiter d'une interface clef en main pour la gestion de comptes dans votre application, commencez par ajouter le package :

```
1 meteor add accounts-ui
```

Enfin, éditez le HTML là où vous souhaitez voir apparaître les contrôles de login :

```
1 <body>
2   <div class="myNavbar">
3     <div class="title">Mon Application
          </div>
4     <div class="login">
5       {{> loginButtons}}
6     </div>
7   </div>
8 </body>
```

Et voilà, c'est aussi simple que ça. Vos utilisateurs peuvent ainsi créer un compte dans votre application et s'y connecter. Le comportement de ce widget est contrôlable via la fonction Accounts.ui.config(). Elle prend en paramètre un objet avec différentes clefs. Celle qui vous intéressera probablement le plus est passwordSignupFields qui vous permet de contrôler les éléments facultatifs et obligatoires lors de l'enregistrement d'un utilisateur. Pour rendre le nom d'utilisateur et l'email obligatoires vous pourrez procéder ainsi, côté client :

```
1 Accounts.ui.config({
2   passwordSignupFields: '
        USERNAME_AND_EMAIL'
3 });
```

Les autres valeurs de configuration possibles sont USERNAME_AND_OPTIONAL_EMAIL (nom d'utilisateur obligatoire, email facultatif), USERNAME_ONLY (uniquement un nom d'utilisateur) et EMAIL_ONLY (uniquement un email).

10 Aller plus loin

Nous avons vu les bases de Meteor et j'espère que vous commencez à entrevoir le potentiel de cette plateforme. Une fois que vous en maîtrisez les concepts fondamentaux, vous êtes capables de produire une application en un temps record. Pour cela, vous pouvez également vous appuyer sur les créations de la communauté qui a déjà publié un certain nombre de packages tiers, destinés à vous simplifier la vie.

La dernière étape dans la production de votre application sera son déploiement. Là encore, Meteor peut vous aider à obtenir un résultat très rapidement, si vous consentez à utiliser l'infrastructure de l'équipe Meteor et un sous-domaine associé.

10.1 Déployer son application

Une fois que votre application sera prête, vous voudrez certainement la rendre publique. Il y a plusieurs moyens de procéder, mais je ne vais parler que de l'option la plus rapide et la plus simple. L'équipe de Meteor met à disposition son infrastructure pour déployer votre application sur un sous-domaine de meteor.com. Pour cela il y a la commande meteor deploy :

```
meteor deploy mon-application.meteor.com
```

Il faut évidemment que le sous-domaine ne soit pas déjà utilisé. La première fois que vous déployez une application on vous demandera d'entrer une adresse email pour vous créer un compte développeur. On vous enverra alors un email pour activer votre compte, choisir un identifiant et un mot de passe. Une fois ceci fait, vous pourrez utiliser la commande meteor login pour vous connecter avec votre compte développeur. Lorsque vous êtes connecté, vous pouvez déployer votre application avec meteor deploy.

Il vous est possible d'autoriser d'autres développeurs à déployer votre application et accéder à ses données avec la commande meteor authorized. La commande suivante liste les développeurs autorisés pour votre application :

```
meteor authorized mon-application
```

Vous pouvez ajouter un nouvel utilisateur de la manière suivante :

```
meteor authorized mon-application --add
    pierre
```

De la même manière vous pouvez utiliser meteor authorized <application> –remove <username> pour supprimer l'autorisation d'un utilisateur. Sachez que le fait d'autoriser un utilisateur lui confère les mêmes droits que vous. Il pourra ainsi ajouter et supprimer d'autres utilisateurs.

11 Conclusion

Meteor est une plateforme qui a pour objectif de simplifier au maximum le développement d'une application Web. Son orientation temps réel fait qu'elle est parfaitement adaptée au développement d'une application Web monopage. Vous n'avez pas à vous soucier des moyens techniques à mettre en oeuvre pour faire communiquer votre serveur et vos clients. Meteor le fait pour vous. Vous pouvez vous concentrer sur l'essentiel et ça, c'est important. Si vous ne souhaitez pas l'adopter pour une application de production, elle peut toujours vous servir pour du prototypage. Tester un concept n'aura jamais été aussi rapide. Ce qui est sûr, c'est que Meteor a des arguments solides pour se faire une place dans le monde des frameworks Node.js.

12 Ressources utiles

Voici un certain nombre de liens qui m'ont été utiles pour la rédaction de ce tutoriel et qui peuvent vous aider dans l'apprentissage de Meteor :

- https ://www.meteor.com/ : lien 33
- https ://www.discovermeteor.com/ : lien 34
- https ://www.eventedmind.com/ : lien 35
- http ://meteorhacks.com/ : lien 36
- http ://www.meteorpedia.com/ : lien 37

Retrouvez l'article de ***Sören Ohnmeiss*** *en ligne : lien 38*

Créer un site pour toutes les plates-formes : extraits du livre

Découvrez ci-dessous les premiers extraits du livre Créer un seul site pour toutes les plates-formes - Aux sources des approches responsive et adaptative aux éditions Eyrolles : lien 39.
Pourquoi développer plusieurs versions d'un site web pour les mobiles, les ordinateurs (portable ou de bureau), voire des applications dédiées ? Il est possible de concilier les usages autour d'un seul site et d'une seule URL : c'est ce que propose l'approche One Web, en respect avec les standards du Web.

1 Introduction

Le web évolue drastiquement ces dernières années. Il a été poussé par la popularisation et la montée en puissance des smartphones. Les réseaux mobiles sont de plus en plus performants, et nous sommes de plus en plus connectés. Accéder à Internet n'a jamais été aussi facile avec les réseaux WiFi et 3G/4G. L'e-commerce et les services en ligne sont en plein essor.

Au début de la démocratisation de la smart-mobilité, les erreurs stratégiques de positionnement et de choix techniques furent nombreuses. Aucune pérennité n'était alors envisageable dans un environnement aussi concurrentiel où l'innovation est une question de survie. Parier sur des technologies émergentes peut s'avérer très gratifiant, ce qui justifie la prise de risque. Cependant, à se concentrer sur la technologie, on oublie parfois que la véritable valeur se trouve dans le contenu ou le service proposé. Les périphériques connectés sont des vecteurs de diffusion et il s'agit d'en extraire leur potentiel maximum pour présenter ce contenu et cette interaction de la manière la plus efficace et attrayante possible.

Aujourd'hui, le marché est installé, les clients équipés. Les constructeurs sont tenaillés entre leur inextinguible besoin de faire évoluer l'offre et les exigences croissantes de leurs clients quant au nombre d'applications disponibles et à leur rétrocompatibilité. Ce contexte est propice à la standardisation. Parallèlement, les développeurs web doivent faire face à la multitude d'appareils et de systèmes présents sur le marché. Pour parvenir à être présents partout, ils s'aident de procédures de tests de plus en plus rigoureuses et d'un outillage logiciel de plus en plus sophistiqué.

Difficile pour une entreprise ou un particulier d'être présent partout à la fois. Le virage de la mobilité était à peine amorcé que la révolution mobile est déjà passée en seconde phase : l'heure est au multi-écran. Les périphériques intermédiaires et hybrides viennent déjà combler les vides laissés par cette migration des usages.

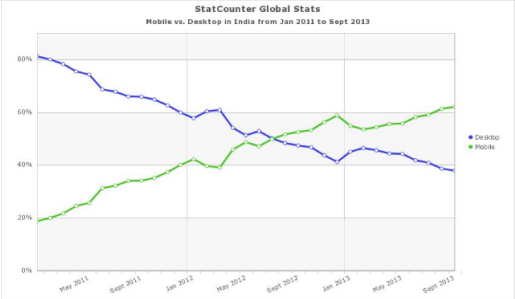
Cet ouvrage a pour objectif de prendre du recul sur le boom des appareils connectés au web en vous donnant les moyens techniques de parvenir avec votre site web à aborder tous ces différents contextes d'utilisation tout en vous préparant à ceux à venir.

2 Contexte et enjeux

Ces dix dernières années, les équipements connectés se sont multipliés pour faire partie intégrante de notre vie quotidienne. De nouveaux types de périphériques et de nouvelles plates-formes système sont apparus et se sont développés très vite. Avant de se lancer dans la création d'une nouvelle application ou d'un nouveau service en ligne, il est essentiel de faire le point sur le contexte actuel.

À présent il n'est plus question de concevoir un site web en négligeant complètement l'usage mobile. Toutes les statistiques de consultation des grands sites sont formelles : la part de clients « mobiles » ne cesse d'augmenter de manière exponentielle. Le phénomène est particulièrement prononcé pour les réseaux sociaux et les sites d'actualité ou de météo. Facebook a annoncé en juillet 2013 disposer de 819 millions d'utilisateurs mobile actifs (au moins une connexion par mois), soit 71 % du total d'utilisateurs actifs sur le réseau social (lien 40) . Le marché mobile comptait en 2012 plus d'un milliard de smartphones en circulation soit un quart de tous les téléphones actifs (lien 41).

La révolution mobile se retrouve globalement dans tous les pays. Mais ce sont avant tout les pays émergents qui sont responsables de cette forte accélération de l'usage mobile. En effet, ce sont dans ces pays que provient la grande majorité des nouveaux internautes. Ceux-ci s'équipent en priorité de smartphones et de tablettes au détriment des ordinateurs fixes et ordinateurs portables. Ainsi, en Inde, le trafic Internet est majoritairement issu de l'usage mobile depuis mai 2012 d'après StatCounter (lien 42).

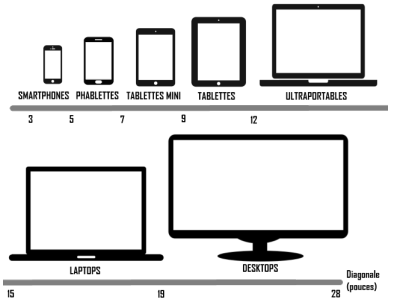


Répartition en Inde du trafic web global entre terminaux mobiles et fixes

Mais ne nous y trompons pas : l'usage mobile n'a pas vocation à remplacer l'usage *desktop*. Certes, le marché du PC a beaucoup chuté en 2013, mais principalement parce que les foyers sont déjà bien équipés. En Europe, les statistiques de consultation des grands sites montrent que le desktop reste et restera sans doute encore pour plusieurs années la classe d'appareil la plus utilisée pour surfer sur le Web. Néanmoins, la part du mobile est suffisamment importante pour que l'on ne puisse plus les ignorer. Il est important de considérer les smartphones non pas comme les successeurs des ordinateurs traditionnels mais plutôt comme un usage complémentaire ancré dans notre quotidien.

Cette logique de complémentarité transparaît clairement avec l'apparition et les bonnes ventes de périphériques intermédiaires. La tablette tactile a ainsi bénéficié d'un grand regain d'intérêt de la part du grand public à la sortie de l'iPad en 2010. Elle se place en juste milieu des usages entre le smartphone et le *laptop*. La plupart des acheteurs déclarent l'utiliser dans leur canapé ou pendant les voyages, quand le laptop est trop encombrant et le smartphone trop petit pour être confortable lors d'une utilisation prolongée ou pour des usages multimédia. On pourrait croire qu'il s'agit d'un cas d'usage minoritaire, et pourtant il a suffi à convaincre les consommateurs. Ce sont 45 millions de tablettes qui ont été livrées au deuxième trimestre 2013 à travers le monde (Source : IDC Worldwide Tablet Tracker, August 5, 2013). En comparaison, sur ce même trimestre ont été livrés 75,6 millions de PC et 238 millions de smartphones.

D'autres périphériques intermédiaires apparaissent également dans les magasins. Sont proposés des claviers en périphérique externe à connecter aux tablettes ; des tablettes dites « hybrides » qui sont en fait des ordinateurs portables dont l'écran est escamotable ; des ordinateurs portables dits « transformables » ou « convertibles » dont l'écran peut se replier grâce à des charnières pour adopter un format tablette. Du côté des téléphones, il s'en vend avec des écrans toujours plus grands, parfois accompagnés d'un stylet. Pour les écrans supérieurs à 5 pouces de diagonale, on les dénomme « phablettes », mot-valise composé de « phone » et « tablette ». Pour les tablettes on observe la tendance inverse : le format 7 pouces de diagonale devient ainsi de plus en plus populaire. Résumons tout cela avec une illustration.



La variété des périphériques, du smartphone à l'ordinateur de bureau

Ceci montre clairement que la révolution mobile est passée et que l'heure est aujourd'hui à la continuité d'usage. Les foyers français sont suréquipés et disposent d'ores et déjà d'un grand nombre d'écrans : 6,3 en moyenne d'après une étude AFP de février 2013. 74% des foyers possèdent à la fois un téléviseur, un ordinateur (desktop ou laptop) et un téléphone mobile. Et si l'on sélectionne ceux dont le chef de foyer a entre 25 et 50 ans, ce chiffre monte à 9 écrans en moyenne à la maison.

Cet engouement pour l'équipement high-tech s'accompagne naturellement d'exigences de plus en plus strictes de la part des utilisateurs en ce qui concerne les applications et les services. Les consommateurs choisiront de préférence des services dont ils peuvent profiter depuis le plus grand nombre de leurs équipements. Ils s'attendent à pouvoir continuer à utiliser un service en passant d'un équipement à un autre ou en renouvelant leur matériel. C'est pourquoi il devient très préjudiciable de ne pas être en mesure de proposer un service grand public sur certaines classes d'appareils ou de systèmes.

Et qu'en est-il des nouveaux périphériques connectés à venir ? Les tiroirs débordent d'idées d'inventeurs. Les montres et les lunettes connectées sont déjà là, avec des produits comme Google Glass ou Samsung Galaxy Gear. Le vêtement connecté ou « cloud clothing » occupe beaucoup les départements Recherche & Développement de plusieurs grandes compagnies. Et ce sont tous les équipements de la maison (domotique, cuisine, électroménager) qui se connectent peu à peu au réseau Internet du domicile. Tous ces nouveaux usages restent encore à déterminer, mais il faut néanmoins les anticiper et se tenir prêts.

C'est dans ce contexte à la fois prospère et intransigeant que se retrouvent aujourd'hui les fournisseurs de contenu et de services sur Internet. Alors de quelles solutions disposent-ils pour être présents sur tous ces appareils afin de s'assurer une bonne visibilité et un maximum d'utilisateurs ? Cet ouvrage vous en présente une : l'approche One Web, un site web unique et ubiquitaire.

3 L'approche One Web

La fracture technologique introduite par la révolution mobile a donné naissance à ce qu'on a nommé le « Web mobile ». L'approche One Web tente de recoller les morceaux.

Pour être présent sur tous les fronts numériques, plusieurs stratégies d'approche ont émergé. Certaines ont été plus populaires que d'autres selon leur temps. Nous allons passer en revue les trois principales puis j'expliquerai pourquoi avoir sélectionné la dernière, l'approche One Web, pour cet ouvrage.

3.1 Un site web desktop et des applications natives pour les smartphones et tablettes

Par application native, on entend une application développée spécifiquement pour une plate-forme, souvent dans un langage et via un kit de développement (*SDK*) qui lui est propre. Les entreprises ont largement fait ce choix au début de la démocratisation des smartphones, entre 2007 et 2010, pour porter leurs services existants vers la mobilité. Si un service disposait d'un bon site web abouti et fiable pour un usage desktop, il pouvait être considéré comme fastidieux et peu prudent de vouloir adapter ce site web pour un usage mobile. Développer une application native avec le SDK de la plate-forme ciblée permet de repartir de zéro et de repenser complètement le service spécifiquement pour cette plate-forme.

Cependant, ce choix de stratégie est beaucoup moins évident aujourd'hui. En effet, celle-ci impose de développer autant d'applications natives que de plates-formes ciblées. Malgré la domination actuelle d'Android, il serait dommageable d'exclure les autres plates-formes comme iOS, Windows Phone ou BlackBerry OS. Et ce sont autant d'applications qu'il faudra gérer, tester et maintenir. Se pose aussi le problème de la fragmentation de versions : tous les téléphones ne font pas tourner la même version du système, et il n'est pas rare de devoir faire évoluer les applications à l'arrivée d'une nouvelle version majeure d'un système. Rappelons également qu'il s'agit d'un marché extrêmement concurrentiel et instable, et que l'on ne peut pas raisonnablement se baser sur les parts de marché actuelles pour parier sur le nombre d'utilisateurs d'une plate-forme les années suivantes. Preuve en est avec la croissance phénoménale d'Android et la chute de BlackBerry OS.

Les constructeurs mettent en avant leurs magasins d'applications et la grande facilité avec laquelle l'utilisateur peut ajouter et conserver un service dans son téléphone. En 2007, l'iPhone d'Apple était de loin le modèle le plus populaire. Or, les applications téléchargées s'affichent sur l'écran d'accueil sur iOS. Disposer d'un affichage du service sur l'écran d'accueil des téléphones des utilisateurs est très séduisant et considéré comme une aubaine pour les équipes marketing.

3.2 Un site web version desktop et un site web version mobile

Précéder l'URL d'un site par « m. » pour un accès optimisé mobile, ou changer l'extension du nom de domaine par « .mobi » cela vous rappelle peut-être quelque-chose. L'idée est de proposer une copie d'un site web existant adapté aux petits écrans et au tactile, avec un contenu et un style extrêmement épurés. Ce site web mobile, bien que distinct du site d'origine en de nombreux points, est souvent rattaché au même nom de domaine et utilise le même *back-end* que son grand frère desktop.

Avant les smartphones, la plupart des téléphones mobiles disposaient de navigateurs utilisant le protocole WAP. Développer un site distinct à ce format était alors la seule option envisageable. Quand les navigateurs mobiles furent suffisamment évolués pour interpréter du HTML 4, cette approche de site distinct ne fut pas abandonnée pour autant. En effet, les résolutions d'écran restaient alors trop faibles et les téléphones trop peu performants pour qu'on leur présente les sites desktop.

Pour guider l'utilisateur vers cette version spécifique au mobile, on détecte l'usage d'un téléphone mobile par différents moyens en JavaScript lors de la connexion au site. L'utilisateur se voit alors proposer de basculer sur la version mobile du site. Dans le pire des cas, il n'a parfois pas le choix, la redirection étant automatique. Or aujourd'hui, il n'est pas rare de voir un utilisateur de smartphone préférer consulter la version desktop jugée plus jolie, plus riche en contenu et davantage maintenue que la version mobile. Les navigateurs web des smartphones sont maintenant extrêmement perfectionnés, et proposent de nombreuses fonctions comme le zoom assisté pour rendre plus facilement consultable les sites prévus à l'origine pour un affichage desktop.

Parallèlement, les périphériques intermédiaires sont venus semer la zizanie dans cette approche. Utilisateur d'une phablette, quelle version suis-je censé choisir ? Il ne faut plus raisonner en fractionnant mobile et fixe à l'ère de la continuité d'usage. Il existe certes encore un certain nombre de téléphones mobiles basiques pour lesquels ces versions mobiles seront davantage adaptées, mais il s'agit d'une proportion trop faible et trop décroissante pour que développer une version spécifique au mobile vaille encore le coup. C'est pourquoi ce genre de sites mobiles est progressivement abandonné et remplacé par des refontes des sites Web desktop d'origine pour essayer de mieux s'accorder à la consultation mobile. Ce qui

nous amène à l'approche One Web, sur laquelle est basé cet ouvrage.

3.3 Un site web unique conciliant tous les usages

Oui, il est aujourd'hui possible de concevoir un et un seul site web proposant une expérience utilisateur convaincante pour toutes les classes de périphériques vues précédemment.

D'aucuns diront que cette promesse a déjà été faite par le passé et qu'il y a eu de quoi être désappointé. Le choix du Web a souvent été critiqué pour les problèmes de performance. Pourtant, les différents navigateurs se sont livrés à la course aux performances pour tenter de convaincre le public par les chiffres. Les moteurs de rendus ont été optimisés et peuvent exploiter l'accélération matérielle pour composer et dessiner plus rapidement les pages web. Entre 2008 et 2013, Mozilla, Google et IE ont tous annoncé des gains énormes pour leurs moteurs d'exécution JavaScript : entre 500 et 2 000 % selon les *benchmarks*.

Si JavaScript en tant que langage interprété ne sera jamais en mesure de rivaliser avec les autres langages compilés, cette différence est devenue suffisamment mince pour ne plus justifier à elle seule les ralentissements de certains sites web. Dans la majorité des cas, la faute revient davantage à de gros défauts d'optimisation de la part des développeurs ; JavaScript est un langage de script simple et haut niveau qui n'encourage pas spécialement à l'optimisation, surtout avec l'abstraction apportée par des frameworks comme jQuery.

Outre les gains de performance des moteurs de rendus et moteurs d'exécution JavaScript, la situation s'est considérablement améliorée depuis cinq ans grâce à plusieurs facteurs :

- les améliorations des navigateurs en support, conformité des standards et en ergonomie ;
- les nombreux ajouts aux langages structurants du Web, HMTL, CSS et JavaScript, que vous aurez l'occasion de découvrir dans les prochains chapitres ;
- l'évolution des réseaux mobiles avec la 3G/3G+/4G, ainsi que la connectivité WiFi intégrée directement à un nombre croissant de périphériques ;
- la montée en puissance du hardware des téléphones mobiles.

Tout l'intérêt de cette approche est bien sûr d'augmenter le nombre d'utilisateurs ciblés tout en centralisant la gestion et le contenu au sein d'un seul service. Un seul site à maintenir et à faire évoluer, c'est l'assurance d'une homogénéité et donc d'une plus grande maîtrise de la qualité de service. Avoir un seul point d'entrée permet également aux consommateurs de plus facilement identifier et partager un service.

Mais concilier les usages, qu'est-ce que cela signifie ? Avant tout il est essentiel de préciser qu'un site web unique n'est pas un site web uniforme. Les usages diffèrent selon la classe de périphérique utilisé par vos clients, car le contexte d'utilisation n'est pas le même. Il s'agit d'adapter le site à ces changements de contexte : c'est ce qu'on appelle la contextualisation. On peut faire une analogie avec une chaîne de magasins disposant de grands hypermarchés et de petites boutiques. Les clients retrouveront la même identité visuelle et la même qualité de service. Mais la taille, l'organisation et le contenu des rayons seront adaptés au contexte, car un client ne se rend pas dans une petite boutique ou dans un hypermarché avec les mêmes intentions.

Cette idée de concilier les usages autour d'un seul site, d'une seule URL, c'est ce qu'on appelle l'approche One Web. Mais ce nom « One Web » est en soi un pléonasme. Car le Web a été créé dans ce sens : pour rendre accessible universellement du contenu. Ce qu'on a appelé « Web mobile » n'a aucun sens pratique. Les mobiles ont aujourd'hui accès au même web que les ordinateurs fixes, il s'agit des mêmes langages, des mêmes protocoles, des mêmes URL. C'est donc juste un jeu d'esprit qui a fragmenté le Web en deux au mépris de la philosophie dans laquelle le Web a été inventé.

The primary design principle underlying the Web's usefulness and growth is universality. (...) it should be accessible from any kind of hardware that can connect to the Internet : stationary or mobile, small screen or large. (Citation extraite de Long Live the Web : A Call for Continued Open Standards and Neutrality ; Scientific American Traduction ; Le premier principe de conception qui sous-tend l'utilité et la croissance du Web est l'universalité. (...) il devrait pouvoir être accessible depuis n'importe quelle sorte de matériel connecté à Internet : fixe ou mobile, avec un écran petit ou grand.)

What's very important from my point of view is that there is one web ... Anyone that tries to chop it into two will find that their piece looks very boring. (Citation extraite de l'interview « US backing for two-tier internet », BBC News, 7 septembre 2007 ; le contexte ne s'appliquait pas au Web mobile mais à la discrimination de contenus, cependant l'idée reste la même Traduction : Ce qui est très important de mon point de vue, c'est qu'il y a un seul web. Quiconque essaie de le couper en deux trouvera que son morceau semble très ennuyeux.)

Tim Berners Lee, inventeur du World Wide Web

Illustration créée à l'occasion du One Web Day qui a lieu chaque 22 Septembre

Derrière cette approche conceptuelle, il existe différentes approches méthodologiques pour permettre aux développeurs de s'adapter de manière pratique à ce mode de pensée. La plus connue est le concept « Mobile First » inventé par Luke Wroblewski, ancien architecte design chez Yahoo. Il consiste à concevoir son site en le pensant prioritairement pour un usage mobile. Ainsi, des efforts supplémentaires sont faits en matière de performance, d'épuration de contenu et de flexibilité. Et pour les clients desktop, selon l'adage « qui peut le plus, peut le moins », l'expérience reste satisfaisante. On pourra également pour ces derniers venir rajouter du contenu et des éléments plus complexes en suivant un principe d'amélioration progressive.

Cette méthodologie a pour avantage d'aller à contre-courant des vieilles méthodes, de forcer les designers et développeurs à sortir de leur cadre habituel pour leur imposer les enjeux et contraintes de l'usage mobile. Néanmoins, elle ne fait que retourner le problème au lieu de le résoudre. Car en pensant de cette manière, on continue à fragmenter le Web, à se confronter aux mêmes obstacles, à ne pas savoir quoi faire avec les périphériques intermédiaires.

La première chose à faire pour bien comprendre l'approche One Web est de ne plus se concentrer sur le conteneur mais sur le contenu. Puis d'avoir conscience que ce contenu, quel qu'il soit, sera amené à être présenté sous des formats, situations et environnements très divers que vous ne pourrez pas tous appréhender. Plutôt que de chercher un contrôle total du rendu de votre site, présentez-le sous une forme simple puis agrémentez-le de petits plus qui amélioreront globalement l'expérience utilisateur.

4 Continuer la lecture

Retrouvez la fiche du livre sur developpez.com (lien 43) et sur le site d'Eyrolles (lien 44).

Une critique du livre par vermine est également disponible ici : lien 45.

*Retrouvez l'article de **Sylvain Pollet-Villard** en ligne : lien 46*

Faire un gestionnaire de contacts avec GoogleMaps et le mkFramework

Dans ce tutoriel, je vous propose d'intégrer facilement un module Google Maps dans votre application web.

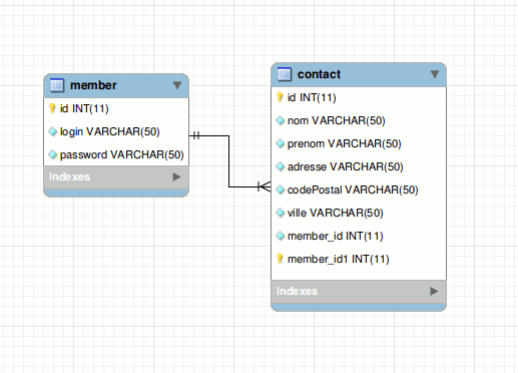
1 Introduction

Ce tutoriel se base sur le framework PHP mkframework, vous pouvez lire une présentation de ce dernier ici : lien 47.

Nous créons une simple application de contact avec un accès restreint afin que chaque utilisateur puisse se créer un compte, puis administrer ses contacts en les géolocalisant via l'API de Google.

2 Le modèle de données

2.1 Voici le schéma MCD



Ci-dessous, le schéma de la base de données que nous allons utiliser :

2.2 Voici la requête SQL

Créez une base de données « googleMapDb » :

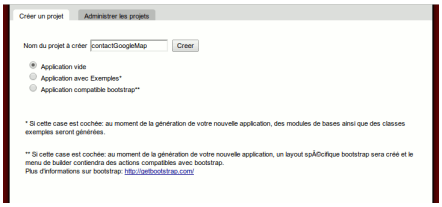
```
1  CREATE TABLE `contact` (
2  `cont_id` int(11) NOT NULL
       auto_increment,
3  `cont_lastname` varchar(50) NOT NULL,
4  `cont_firstname` varchar(50) NOT NULL,
5  `cont_adress` varchar(50) NOT NULL,
6  `cont_zip` varchar(50) NOT NULL,
7  `cont_city` varchar(50) NOT NULL,
8  `cont_picture` varchar(100) NOT NULL,
9  `memb_id` int(11) NOT NULL,
10 PRIMARY KEY  (`cont_id`)
11 );
12 CREATE TABLE `member` (
13 `memb_id` int(11) NOT NULL
       auto_increment,
14 `memb_username` varchar(50) NOT NULL,
15 `memb_password` varchar(50) NOT NULL,
16 PRIMARY KEY  (`memb_id`)
17 );
```

Le champ « mot de passe » doit faire au minimum 40 caractères, car il stockera le mot de passe hashé avec la fonction sha1() de PHP. Plus d'informations sur la fonction sha1 ici : lien 48.

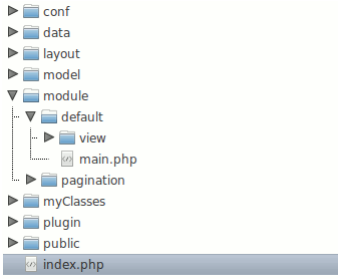
Si vous utilisez une autre fonction de hashage, pensez à modifier la longueur de ce champ en conséquence.

3 Création de l'application

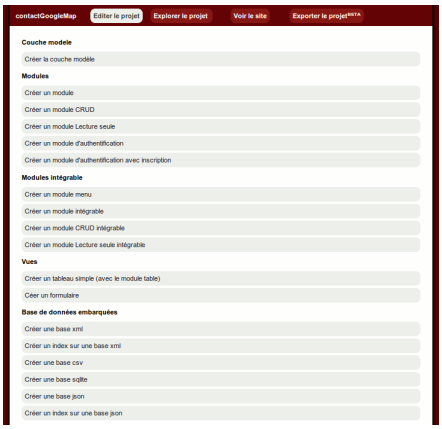
Rendez-vous à l'adresse du framework, saisissez « contactGoogleMap » et validez :



Voici l'arborescence créée :



Une fois l'application générée par le builder, vous êtes redirigé sur la page d'administration du projet :



4 Éditez le fichier de configuration des connexions

Pour gérer les connexions de votre application, vous avez à votre disposition un fichier de configuration dans le répertoire *conf/*.

Éditez le fichier *conf/connexion.ini.php* pour paramétrer votre serveur MySQL ainsi :

```
1 ;<?php die()?>
```

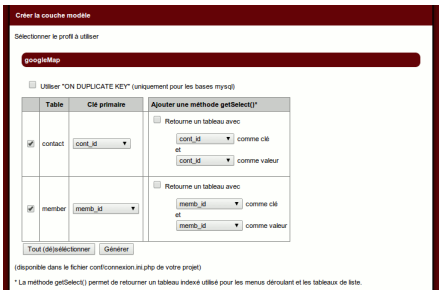
```
2 [db]
3 googleMap.dsn="mysql:dbname=googleMapDb;
    host=localhost"
4 googleMap.sgbd=pdo_mysql
5 googleMap.username=root
6 googleMap.password=root
```

5 Créez la couche modèle

Cliquez sur « créer la couche modèle ».

Vous voyez la liste des profils de connexion précédemment renseignés dans le fichier :

Cliquez sur le profil « googleMapDb » :

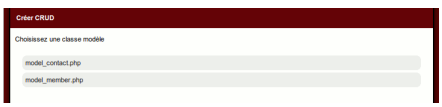


Vous voyez la liste des tables visibles par le profil de connexion sélectionné.

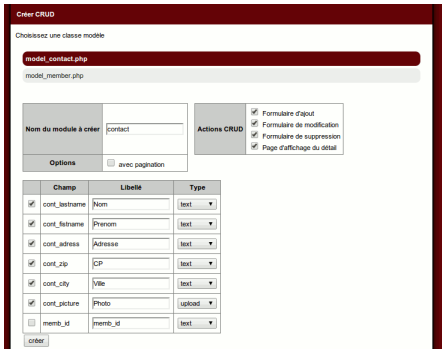
En validant, le builder vous crée un fichier de classe par table.

6 Créez le module contact

Cliquez sur « créer un module CRUD » :



Vous voyez la liste des classes modèle de l'application précédemment créée. Une classe pour les membres et une autre pour les contacts. Sélectionnez la classe « model_contact.php » :



Lorsque vous sélectionnez votre classe modèle contact, vous voyez apparaître un tableau listant les champs présents dans la table « contact ».

Le builder vous permet de personnaliser le CRUD qui sera généré : vous pouvez choisir de créer ou non les pages d'ajout, de modification et de suppression.

Vous pouvez également préciser les libellés des champs tels qu'ils seront affichés en entête de tableau et nom de champ de formulaire.

Pensez à modifier le champ « image », sélectionnez dans le menu déroulant « upload » afin de permettre d'uploader une photo de votre contact.

Validez le formulaire, le builder vous générera un module contact dans le répertoire module de votre application :

Le builder vous détaille les répertoires et les fichiers créés.

De plus, il vous propose un lien pour consulter le rendu de votre module.

7 Ajoutez l'authentification

Notre application doit permettre de s'inscrire pour ensuite administrer ses contacts à partir de leur position sur GoogleMap. Nous allons ici simplement ajouter la partie « authentification » plus « inscription ».

Cliquez sur « Créer un module d'authentification avec inscription » :

Le builder vous indique que l'application nécessite des modifications pour pouvoir créer ce module d'authentification.

Il vous invite à créer deux méthodes dans la classe modèle gérant les comptes de connexion ; dans notre cas, la classe member.

Copiez ces deux méthodes et collez-les dans le fichier *model/model_member.php* dans la classe model_member.

Pour la première méthode, il faut bien remplacer les champs « login » et « pass » par le nom des champs équivalents dans notre table member, ici « memb_username » et « memb_password ».

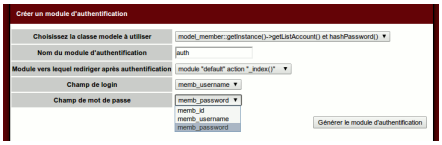
Ce qui donnera pour notre classe *model_member.php* :

```
1  <?php
2  class model_member extends
       abstract_model{
3
4      protected $sClassRow='row_member';
5
6      protected $sTable='member';
7      protected $sConfig='googleMap';
8
9      protected $tId=array('id');
10
11     public static function getInstance()
           {
12         return self::_getInstance(
               __CLASS__);
13     }
14
15     public function findById($uId){
16         return $this->findOne('SELECT *
               FROM '.$this->sTable.' WHERE
                id=?',$uId );
17     }
18     public function findAll(){
19         return $this->findMany('SELECT *
                FROM '.$this->sTable);
20     }
21
22     public function getListAccount(){
23
24         $tAccount=$this->findAll();
25
26         $tLoginPassAccount=array();
27
28         if($tAccount){
29             foreach($tAccount as
                   $oAccount){
30                 //on crée ici un tableau
                       index par nom d'
                      utilisateur et mot
                      de pase
31                 $tLoginPassAccount[
                       $oAccount->
                       memb_username][
```

```
                    $oAccount->
                    memb_password]=
                    $oAccount;
32              }
33          }
34
35          return $tLoginPassAccount;
36
37      }
38
39      public function hashPassword(
            $sPassword){
40          //utiliser ici la méthode de
                votre choix pour hasher
                votre mot de passe
41          return sha1('monSel1234'.
                $sPassword);
42      }
43  }
```


Pour la méthode hashPassword(), je vous invite à ajouter un « sel » afin d'améliorer la sécurité.

Une fois la classe model_member modifiée, réactualisez la page de création de module d'authentification :

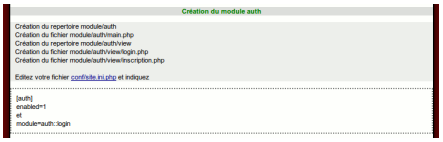


Cette fois-ci, le builder vous invite à renseigner plusieurs choses :

- la classe modèle à utiliser (notre classe model_member) ;
- le nom du module d'authentification à créer ;
- le module vers lequel rediriger, une fois connecté ;
- le champ de login et celui de mot de passe.

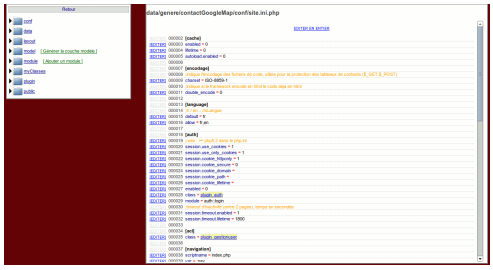
Renseignez pour le module à rediriger : « contact : :index » et pour les champs de login et mot de passe « login » et « password », puis validez.

Le builder vous créera un nouveau module dans le répertoire du même nom :



Comme vous pouvez le lire, il vous demande d'activer l'authentification en modifiant une variable dans un fichier de configuration.

Il propose également un lien pour vous rendre sur ce fichier de configuration, cliquez dessus :

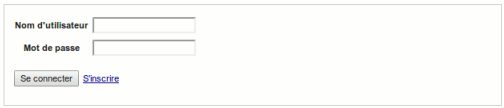


Le builder vous permet de naviguer et modifier vos fichiers dans votre navigateur. Vous n'avez plus qu'à modifier la ligne demandée dans la section [auth] :

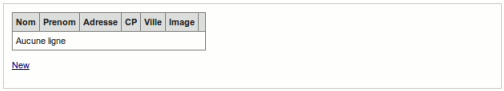
```
1  [auth]
2  enabled=1
```

À partir de ce moment, vous ne pouvez plus accéder au site, il faut vous inscrire.

Cliquez sur « voir le site » pour vous rendre sur votre application, vous êtes automatiquement redirigé sur la page de « login » :



Cliquez sur le lien « s'inscrire » et créez un compte. Identifiez-vous et vous serez redirigé sur le module contact :



8 Modifions le module contact pour lier les contacts au membre connecté

Avant d'enregistrer un contact, nous allons modifier le module contact pour qu'il lie automatiquement les contacts créés au membre connecté, et filtrer l'affichage afin que chaque membre puisse visualiser et éditer uniquement ses contacts.

Modifiez le fichier *module/contact/main.php*.

Avec la méthode processSave() :

```
1  private function processSave(){
2      if(!_root::getRequest()->isPost() ){
           //si ce n'est pas une requete
           POST on ne soumet pas
3          return null;
4      }
5
6      $oPluginXsrf=new plugin_xsrf();
7      if(!$oPluginXsrf->checkToken( _root
           ::getParam('token') ) ){ //on
           verifie que le token est valide
8          return array('token'=>
               $oPluginXsrf->getMessage() )
               ;
9      }
10
11     $iId=_root::getParam('id',null);
```

```
12      if($iId==null){
13          $oContact=new row_contact;
14          //on force ici l'id du membre
                par celui du membre connecté
15          $oContact->member_id=_root::
                getAuth()->getAccount()->id;
16      }else{
17          $oContact=model_contact::
                getInstance()->findById(
                _root::getParam('id',null) )
                ;
18      }
19
20      $tColumn=array('cont_lastname','
            cont_firstname','cont_adress','
            cont_zip','cont_city');
21      foreach($tColumn as $sColumn){
22          $oContact->$sColumn=_root::
                getParam($sColumn,null) ;
23      }
24
25      $tColumnUpload=array('cont_picture')
            ;
26      if($tColumnUpload){
27          foreach($tColumnUpload as
                $sColumnUpload){
28              $oPluginUpload=new
                    plugin_upload(
                    $sColumnUpload);
29              if($oPluginUpload->isValid()
                    ){
30                  $sNewFileName=_root::
                        getConfigVar('path.
                        upload').
                        $sColumnUpload.'_'.
                        date('Ymdhis');
31
32                  $oPluginUpload->saveAs(
                        $sNewFileName);
33                  $oContact->
                        $sColumnUpload=
                        $oPluginUpload->
                        getPath();
34              }
35          }
36      }
37
38      if($oContact->save()){
39          //une fois enregistré, on est
                redirigé (vers la page liste
                )
40          _root::redirect('contact::list')
                ;
41      }else{
42          return $oContact->getListError()
                ;
43      }
44
45  }
```

Nous avons ajouté ici, lors de la création d'un contact, l'assignation du champ « member_id » avec celui de notre utilisateur connecté.

Il faut de plus, filtrer les contacts affichés dans le tableau. Pour cela, modifiez le modèle contact.

Éditez le fichier *model/model_contact.php* pour ajouter une méthode findByMember() :

```
1  public function findByMember($member_id)
       {
2      return $this->findMany('SELECT *
           FROM '.$this->sTable.' WHERE
           memb_id=?',$member_id);
3  }
```

Et modifiez en conséquence l'appel dans le fichier *module/contact/main.php*.

Remplacez dans la méthode Remplacez dans la méthode _list() :

```
1  $tContact=model_contact::getInstance()->
       findAll();
```

par

```
1  $tContact=model_contact::getInstance()->
       findByMember(_root::getAuth()->
       getAccount()->id);
```

Saisissez un ou deux contacts avec leur adresse.

Supprimez également l'affichage de l'adresse de la photo uploadée :

Éditez la vue *module/contact/view/list.php* pour supprimer la ligne de l'image.

Ce qui donnera :

```
1  <table class="tb_list">
2      <tr>
3
4          <th>Nom</th>
5          <th>Prenom</th>
6          <th>Adresse</th>
7          <th>CP</th>
8          <th>Ville</th>
9          <th></th>
10     </tr>
11     <?php if($this->tContact):?>
12         <?php foreach($this->tContact as
                $oContact):?>
13         <tr <?php echo plugin_tpl::
               alternate(array('','class="
               alt"'))?>>
14             <td><?php echo $oContact->
                   cont_lastname ?></td>
15             <td><?php echo $oContact->
                   cont_firstname ?></td>
16             <td><?php echo $oContact->
                   cont_adress ?></td>
17             <td><?php echo $oContact->
                   cont_zip ?></td>
18             <td><?php echo $oContact->
                   cont_city ?></td>
19             <td>
20                     <a href="<?php echo
                           $this->getLink('
                           contact::edit',
                           array(
21                         'id'=>$oContact-
                               >getId()
22                     )
23                     )?>">Edit</a>
24                     |
25                     <a href="<?php echo
                           $this->getLink('
                           contact::delete
                           ',array(
26                         'id'=>$oContact-
                               >getId()
27                     )
28                     )?>">Delete</a>
29                     |
30                     <a href="<?php echo
                           $this->getLink('
                           contact::show',
                           array(
31                         'id'=>$oContact-
                               >getId()
```

```
                    )
                    )?>">Show</a>
            </td>
        </tr>
        <?php endforeach;?>
    <?php else:?>
        <tr>
            <td colspan="7">Aucune ligne
```

```
            </td>
        </tr>
    <?php endif;?>
</table>

<p><a href="<?php echo $this->getLink('
    contact::new') ?>">New</a></p>
```

9 Intégration du module GoogleMap dans l'application

Pour le module GoogleMap, il faut vous rendre sur le site du mkframework pour le télécharger.

Rendez-vous à l'adresse suivante : lien 49, rubrique « Télécharger les modules » et téléchargez l'archive.

Sur la page de téléchargement, une description vous indique comment utiliser ce module dans votre site.

Désarchivez-le dans le répertoire *module/* de votre application.

Nous allons ajouter, sous la liste des contacts, une carte de Google indiquant la position de nos contacts avec la possibilité de cliquer dessus pour avoir le détail ainsi qu'un lien pour les éditer.

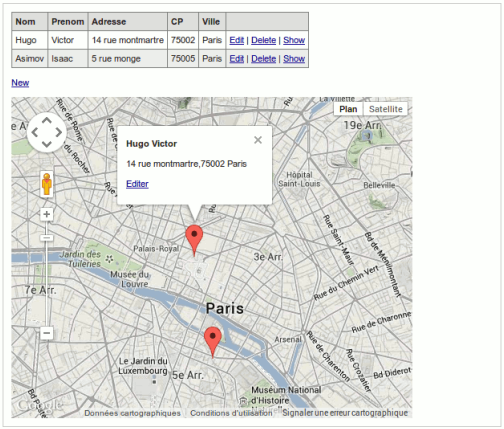
Éditer le fichier *module/contact/main.php*, méthode _list() et en fin de méthode, ajoutez :

```
$oModuleGoogleMap=new module_googleMap()
    ;
$oModuleGoogleMap->setWidth(500);
$oModuleGoogleMap->setHeight(400);
$oModuleGoogleMap->setZoom(13);


if($tContact ){
foreach($tContact as $oContact){
    //un pointer avec un peu de contenu
        html lorsque l'on clique dessus
    $oModuleGoogleMap->
        addPositionWithContent($oContact
        ->cont_adress.', '.$oContact->
        cont_zip.' '.$oContact->
        cont_city, $oContact->
        cont_lastname.' '.$oContact->
        cont_firstname,array(
            '<h1>'.$oContact->
                cont_lastname.' '.
                $oContact->
                cont_firstname.'</h1>',
            '<p>'.$oContact->cont_adress
                .',',
            ''.$oContact->cont_zip.' '.
                $oContact->cont_city.'<
                /p>',
            '<p><a href="'._root::
                getLink('contact::edit'
                ,array('id'=>$oContact-
                >id)).'">Editer</a></p>
                '
     ));
}
}

$this->oLayout->add('main',
    $oModuleGoogleMap->getMap());
```

Ce qui vous donnera :



Vous pouvez également utiliser la fonctionnalité d'upload d'une photo afin d'afficher une photo du contact.

Pour cela, modifiez le code précédent ainsi (pour y ajouter l'affichage de l'image uploadée) :

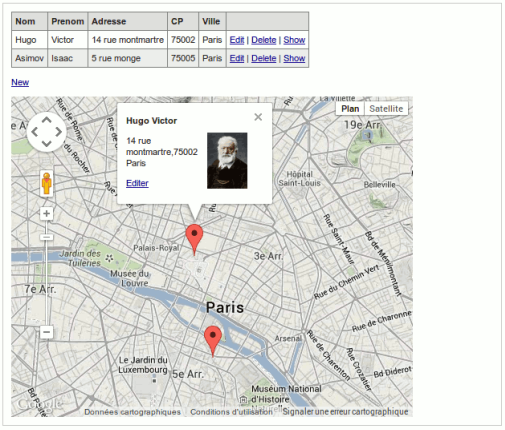
```
$oModuleGoogleMap=new module_googleMap()
    ;
$oModuleGoogleMap->setWidth(500);
$oModuleGoogleMap->setHeight(400);
$oModuleGoogleMap->setZoom(13);


foreach($tContact as $oContact){

    $sImage=null;
    if($oContact->cont_picture!=''){
        $sImage='<img style="float:right
            ;width:50px" src="'.
            $oContact->cont_picture.'"/>
            ';
    }

    //un pointer avec un peu de contenu
        html lorsque l'on clique dessus
    $oModuleGoogleMap->
        addPositionWithContent($oContact
        ->cont_adress.', '.$oContact->
        cont_zip.' '.$oContact->
        cont_city, $oContact->
        cont_lastname.' '.$oContact->
        cont_firstname,array(
            '<h1>'.$oContact->
                cont_lastname.' '.
                $oContact->
                cont_firstname.'</h1>',
            $sImage,
            '<p>'.$oContact->cont_adress
                .',',
            ''.$oContact->cont_zip.' '.
                $oContact->cont_city.'<
                /p>',
```

```
20              '<p><a href="'._root::
                    getLink('contact::edit'
                    ,array('id'=>$oContact-
                    >id)).'">Editer</a></p>
                    '
21      ));
22  }
23
24  $this->oLayout->add('main',
        $oModuleGoogleMap->getMap());
```



10 Conclusion

Vous avez pu apprécier le gain de productivité et la simplicité d'utilisation de ce framework ainsi que de son builder.

J'espère vous avoir donné envie d'approfondir votre intérêt envers ce challenger qu'est le mkframework.

Retrouvez l'article de ***Michael Bertocchi*** *en ligne : lien 50*

AJAX

Les derniers tutoriels et articles

Pourquoi utiliser TypeScript ?

Dans cet article le lecteur trouvera de mauvaises raisons pour écarter TypeScript et de bonnes raisons pour l'adopter.

1 Introduction

« TypeScript, le langage compilé en JavaScript ? Mais pourquoi faudrait-il quitter JavaScript ? »

Le langage JavaScript ne manque pas d'atouts et le mieux est souvent l'ennemi du bien. Délaisser ce qui fonctionne est un risque. Et puis il existe de solides arguments en défaveur des langages construits sur JavaScript, examinons-les :

1. La question de la pérennité des syntaxes non standard ;
2. La problématique de l'intégration avec l'existant ;
3. La difficulté de former ou de trouver les développeurs.

Je répondrai à ces trois questions au fil de l'exposé tout en situant TypeScript par rapport aux technologies concurrentes. Je tenterai ensuite de donner envie au lecteur d'essayer.

2 TypeScript est un choix sans risque

Avant toute chose, TypeScript n'est pas un choix dangereux pour l'équipe de développeurs qui envisage de l'adopter.

En effet, le nouveau langage fait sien le slogan de CoffeeScript : « It's just JavaScript ». Le compilateur TypeScript n'ajoute aucune dépendance à une bibliothèque JavaScript. Voilà qui contraste avec la solution proposée par Traceur (lien 51) dont le code compilé s'appuie massivement sur un *runtime* en JavaScript.

Plus encore, le développeur TypeScript se repère sans effort dans son propre code traduit en JavaScript. En vérité le code JavaScript après compilation est impeccable et pourrait être repris comme base de travail JavaScript s'il fallait un jour quitter le navire.

3 Faire coopérer TypeScript et JavaScript ?

TypeScript est un surensemble de la syntaxe JavaScript. Autrement dit, à peu de choses près, tout code JavaScript compilera en TypeScript.

TypeScript s'intègre alors naturellement dans l'écosystème JavaScript. Les bibliothèques de programmation JavaScript comme jQuery ou Modernizr sont appelées normalement depuis du code TypeScript. Et dans l'autre sens, le code TypeScript, une fois compilé, s'utilise aisément depuis du code écrit directement en JavaScript. Ainsi, les *frameworks* AngularJS et consorts travaillent sans faire d'histoire avec du code TypeScript compilé.

Cette intégration de TypeScript avec le Web existant est à mettre en opposition avec le langage Dart (lien 52) conçu comme un environnement autonome en rupture avec JavaScript et dans lequel tout est à refaire. La syntaxe de TypeScript, quant à elle, est en continuité avec celle du JavaScript. Aucun travail d'adaptation de l'existant n'est à faire et la baisse de productivité due à l'effort d'apprentissage est limitée à quelques jours, surtout si le développeur a déjà de l'expérience dans d'autres langages à objets.

4 La syntaxe TypeScript est pérenne

JavaScript a évolué depuis son invention en 1995. Le langage fut normalisé par Ecma International en 1997 sous le nom de ECMAScript. Internet Explorer 8 par exemple comprend l'ECMAScript 3. Le JavaScript interprété par les navigateurs en 2014 est ECMAScript 5.1.

Certains ajouts syntaxiques apportés par TypeScript sont une implémentation de fonctionnalités prévues dans le prochain ECMAScript 6. Par rapport aux fonctionnalités de ECMAScript 6 (lien 53) listées par Luke Hoban, TypeScript implémente les suivantes : *Arrows*, *Classes*, *Default + Rest + Spread* et *Modules*. Cela fait de TypeScript un langage essentiellement normalisé ce qui, au passage, lui donne l'avantage sur CoffeeScript (lien 54) pour ce qui est de la pérennité.

Cela implique également que le compilateur TypeScript est destiné à être simplifié. Lorsque les navigateurs implémenteront nativement ECMAScript 6, le compilateur se repliera sur les seules fonctionnalités non normalisées de TypeScript.

5 TypeScript et Microsoft ?

TypeScript est porté par Microsoft. Ces dernières années Microsoft a opéré un tournant radical par rapport à l'*open source* et l'on peut s'en réjouir. TypeScript est sous licence Apache 2. En outre le compilateur TypeScript est utilisable en dehors de l'univers Microsoft, il est en effet présent depuis le début sur Node.js.

Deux éléments sont révélateurs de l'importance du projet TypeScript pour Microsoft. En premier lieu JavaScript est devenu un langage prioritaire pour Microsoft. Pour appuyer notre propos, relevons son support en natif par Windows 8, ou encore le fait que JavaScript est une porte de sortie élégante pour VBA (lien 55), le vieux langage de script étant devenu une ornière depuis DotNet.

En second lieu le projet TypeScript est dirigé par Anders Hejlsberg qui n'a fait que du solide : il fut l'architecte de Turbo Pascal, de Delphi puis, chez Microsoft, de DotNet/C#.

Je suis tenté de conclure, mais c'est personnel, que TypeScript est promis à un avenir immense. Et si Microsoft ne communique pas fort sur sa technologie, c'est peut-être que l'éditeur préfère prendre le temps de travailler sur l'intégration du langage et du Web dans ses divers outils ?

... Alors, finalement, TypeScript, pourquoi pas ?

La suite de l'article est consacrée aux apports de TypeScript.

6 La programmation orientée objet et TypeScript

En JavaScript, jusqu'à l'actuel ECMAScript 5, la notion de classe n'existe pas et le développeur travaille directement sur les objets. Un mécanisme peut servir à émuler un comportement de classe : le *prototype* dont l'usage est malheureusement incomplet. Impossible en particulier de déclarer des membres privés dans le prototype. L'encapsulation est certes réalisable au niveau de l'objet par des *closures*, mais il en résulte une empreinte mémoire inutilement large et une durée d'initialisation allongée puisque chaque objet conserve alors ses propres pointeurs vers chacune des méthodes privées.

TypeScript apporte à JavaScript une programmation orientée objet classique.

Entre parenthèses je tiens à insister sur quelque chose de fondamental : les bonnes pratiques en JavaScript, dont les *closures* et la programmation asynchrone par *callbacks*, restent de bonnes pratiques en TypeScript. La syntaxe des classes ne retire rien, elle s'ajoute.

Mais trêve de bavardages. J'invite le lecteur à ouvrir sans attendre le *Playground* de TypeScript (lien 56) et à y tester le code que voici :

```
1  interface SayHello {
2    say(name: string): void;
3  }
4  class MyFirstClass implements SayHello {
5    constructor(private prefix = 'Hello')
         {
6    }
7    public say(name: string) {
8      alert(this.makeMessage(name));
9    }
10   private makeMessage(name: string):
         string {
11     return this.prefix + ', ' + name + '
           !';
12   }
13 }
14 var sayToWorld = function (sh: SayHello)
       {
15   sh.say('World');
16 };
17 sayToWorld(new MyFirstClass());
```

Le code est facile à comprendre. Une interface SayHello est déclarée, une classe MyFirstClass implémentant cette interface vient ensuite, puis une fonction sayToWorld est définie, qui attend une implémentation de l'interface en paramètre. Remarquez le private prefix directement comme argument du

constructeur : la présence de l'accesseur est un raccourci pour déclarer la variable d'instance *prefix* et lui affecter la valeur du paramètre du même nom (lequel reçoit ici 'Hello' en valeur par défaut).

Les fonctionnalités apportées par TypeScript sont décrites dans le *Handbook* (lien 57) et les spécifications (PDF) (lien 58). L'objet de cet article n'est pas d'en faire le tour mais je recommande tout de même au lecteur de vite s'intéresser aux modules qui seront les espaces de nommage dans ECMAScript 6 et le sont déjà dans TypeScript.

Au-delà des classes et des valeurs par défaut venant de ECMAScript 6, TypeScript ajoute le typage en suffixe (d'une manière qui rappelle singulièrement la syntaxe de Hack : lien 59) et les accesseurs private et public. Le mécanisme de déclaration et d'implémentation des interfaces, lié au typage, mérite une attention particulière.

7 Les interfaces TypeScript

Les interfaces en TypeScript sont particulièrement souples. Dans le code donné ci-dessus, retirez la mention implements SayHello au niveau de la déclaration de la classe : le code reste valide. Renommez maintenant la méthode say : le compilateur relève l'erreur au niveau de l'appel à sayToWorld.

Vous pouvez même bénéficier des contrôles de typage tout en optant pour une programmation au niveau de l'objet à la manière classique de JavaScript. Par exemple ce code :

```
1  sayToWorld({
2    'say': function (name) {
3      alert('Hi ' + name + '!');
4    }
5  });
```

Ici la classe n'est pas utilisée, la fonction sayToWorld est appelée avec un objet implémentant directement notre interface.

Notons que les interfaces, tout comme le typage et les accesseurs, n'affectent pas les performances puisqu'elles sont absentes du code JavaScript généré.

8 Typage statique ?

Pourquoi du typage statique ? Pour simplifier la documentation nécessaire à la collaboration entre les développeurs et pour aider l'IDE. Les développeurs TypeScript bénéficient d'une aide à la saisie que leurs homologues en JavaScript leur envieraient s'ils savaient, et ce y compris sur des bibliothèques écrites en JavaScript comme jQuery. Des fichiers de définitions répertoriant de nombreuses API existantes sont en effet maintenus par l'importante communauté de DefinitelyTyped : lien 60.

À savoir : le typage est vérifié au niveau de la compilation. Il est facile de le tromper, par exemple en manipulant dynamiquement les membres d'un objet.

Le typage de TypeScript est optionnel. Il n'est pas là pour restreindre la créativité du développeur et à mon avis ce ne serait pas une bonne pratique de toujours tout typer tout le temps ou encore de laisser de côté le code dynamique. En revanche, déclarer les types dans les méthodes publiques d'une classe servant d'API, ou encore dans les fonctions exportées par un module Node.js, est certainement une pratique à recommander.

TypeScript sait également induire le type à partir d'une déclaration. Par exemple, les deux instructions suivantes sont équivalentes :

```
1  var s = 'Hello';
```

```
1  var s: string = 'Hello';
```

La plus lisible, la première, est sans doute la meilleure. Dans les deux cas la variable est typée et par exemple l'instruction suivante lève une erreur :

```
1  s = 123; // Cannot convert 'number' to '
       string'.
```

Si le développeur souhaite volontairement changer le type d'une variable, le « joker » any fera taire le compilateur :

```
1  var s: any = 'Hello';
2  s = 123; // ok
```

9 Comment démarrer avec TypeScript

Ça se tente ? Voici comment démarrer :

1. Installer Node.js : lien 61 ;
2. En ligne de commande, installer le compilateur TypeScript sur Node.js avec la commande *npm* donnée sur le site officiel : lien 62 ;
3. Il vous faudra un IDE : WebStorm (lien 63) (il embarque une licence d'un mois pour tester), Eclipse avec le plugin pour TypeScript (lien 64), ou bien sûr Visual Studio pour ceux qui l'ont ;
4. Sous WebStorm la procédure est la suivante :

créer un projet puis créer un premier fichier « .ts » avec du code TypeScript ; l'IDE propose alors spontanément d'ajouter un « Watcher » chargé de compiler TypeScript à la volée, il faut accepter puis le configurer afin qu'il ne soit appelé qu'à l'enregistrement : File → Settings → File Watchers → TypeScript (edit) → Décocher « Immediate file synchronization » ;

5. L'apprentissage demandera un peu de lecture en anglais ; rendez-vous sur le *Handbook* du site officiel : lien 65.

Vous êtes lancé, et le forum TypeScript vous attend : lien 66.

Retrouvez l'article de ***Tarh*** *en ligne : lien 67*

2D/3D/Jeux

Les derniers tutoriels et articles

Le guide du débutant à Construct2

Bravo, et bien joué d'avoir choisi Construct 2 ! Commençons à faire votre premier jeu en HTML5. Nous allons faire le jeu « Ghost Shooter » (tueur de fantômes) utilisé en démonstration.

1 Introduction

Bravo, et bien joué d'avoir choisi Construct 2 ! Commençons à faire votre premier jeu en HTML5. Nous allons faire le jeu « Ghost Shooter » (tueur de fantômes) utilisé en démonstration. Essayez-le ici (lien 68) afin de comprendre ce que nous allons réaliser : un joueur qui regarde en direction du curseur de la souris, se déplace à l'aide des flèches du clavier et tire sur des monstres avec la souris. Vous allez apprendre tout ce dont vous avez besoin afin de réaliser un jeu simple depuis les « layers » (couches) jusqu'au « event system » (système d'évènements).

1.1 Tutoriels alternatifs

Il existe une alternative au guide du débutant : How to make a platform game (lien 69), qui vous permettra de faire un jeu de plate-forme de type « jump-and-run » (saute et cours) plutôt qu'un jeu de tir vu de dessus. Vous pouvez débuter avec n'importe quel tutoriel, mais nous vous suggérons fortement de parcourir les deux afin d'avoir une bonne idée de la façon de réaliser ces deux types de jeu !

Il existe aussi How to make an Asteroids clone in under 100 events (lien 70) par Kyatric qui est un peu plus avancé/compliqué mais qui est aussi très détaillé.

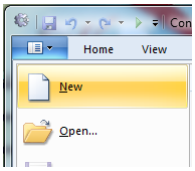
Au moment de la traduction de ce présent tutoriel, les alternatives n'ont pas encore été traduites.

2 Installer Construct 2

Si ce n'est pas déjà fait, téléchargez une copie de la dernière version de Construct 2 ici : lien 71. L'éditeur de Construct 2 ne fonctionne que sous Windows, mais les jeux réalisés peuvent fonctionner n'importe où, aussi bien sous Mac, Linux, iPad ? Construct 2 peut être installé sur un compte d'utilisateur limité (un compte non administrateur sous Windows). Construct 2 est aussi portable, donc vous pouvez l'installer sur une clé USB et l'emporter partout avec vous.

2.1 Commencer avec Construct 2

Maintenant que vous êtes prêts, exécutez Construct 2. Cliquez sur le bouton *File* (fichier) et sélectionnez *New* (nouveau).

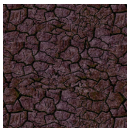


Vous n'avez rien à changer dans la fenêtre *New Project* (nouveau projet), cliquez simplement sur *Create project* (créer un projet). Construct 2 gardera l'intégralité du projet dans un seul fichier d'extension *.capx* pour vous. Vous devriez voir un *layout* vide - l'espace de design où vous créez et positionnez vos objets. Pensez un layout comme un niveau de votre jeu ou l'écran de menu. Dans d'autres outils cela peut être appelé *room* (salle), *scene* (scène) ou *frame* (cadre).

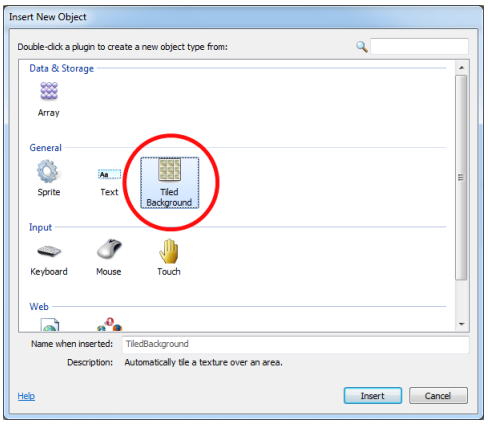
2.2 Insérer des objets

2.2.a Fond en tuiles

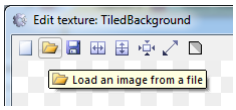
La première chose que nous voulons est un arrière-plan qui se répète. L'objet *Tiled background* peut faire ça pour nous. D'abord, voici une texture de fond. Faites un clic droit et « enregistrer sous » quelque part sur votre ordinateur :



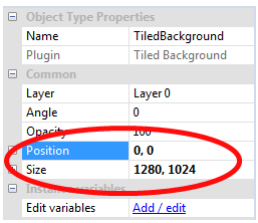
Maintenant **double-cliquez** sur le *layout* pour y insérer un nouvel objet, (plus tard si l'espace est déjà plein, vous pourrez aussi faire un clic droit et sélectionner *Insert new object* (insérer un nouvel objet)). Une fois que la fenêtre Insert new object apparaît, **double-cliquez sur l'objet « *Tiled Background* »** pour l'insérer.



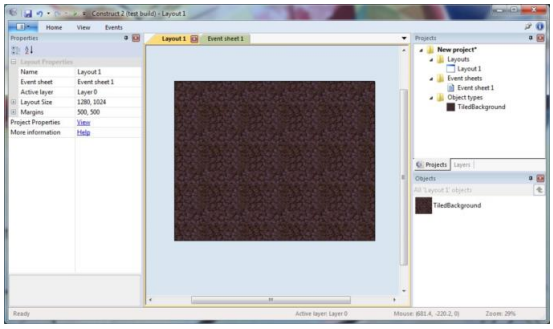
Votre curseur prend la forme d'une croix pour vous indiquer qu'il faut placer l'objet. Cliquez quelque part au milieu du *layout*. L'éditeur de texture (*texture editor*) s'ouvre afin que vous insériez une texture. Importez l'image que vous avez précédemment sauvegardée. Cliquez sur l'icône « dossier » pour charger une texture depuis votre disque, naviguez jusqu'à l'endroit où vous avez sauvegardé le fichier et sélectionnez-le.



Fermez l'éditeur de texture en cliquant sur le X en haut à droite. Si un avertissement apparaît assurez-vous de sauvegarder ! Désormais vous devriez voir votre objet *Tiled Background* dans le *layout*. Redimensionnez-le afin de couvrir l'intégralité du *layout*. Assurez-vous que l'objet soit bien sélectionné, alors la *Properties bar* (barre des propriétés) sur votre gauche devrait vous montrer toutes les propriétés de l'objet, incluant sa taille (*size*) et sa position. Définissez sa position à 0, 0 (le point en haut à gauche du *layout*) et sa taille (*size*) à 1280, 1024 (la taille du *layout*).



Vérifiez votre travail. Maintenez la touche **Ctrl** de votre clavier et faites défiler **la molette de votre souris vers le bas** afin de dézoomer. Vous pouvez aussi cliquer sur *View - zoom out* plusieurs fois dans le bandeau en haut de votre écran pour obtenir le même résultat. Vous pouvez aussi maintenir la touche espace ou le bouton du milieu de votre souris pour déplacer la vue. Joli n'est-ce pas ? Votre objet *Tiled Backround* devrait maintenant recouvrir le *layout* entièrement :



Pressez Ctrl + 0 ou cliquez sur « *View - zoom to 100%* » pour revenir à la vue originale.

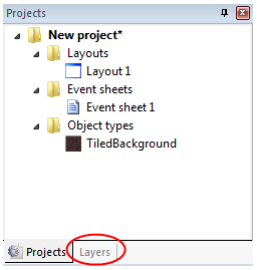
Si vous êtes impatients comme moi, cliquez sur la petite icône *run* (exécuter) dans la barre de titre, un navigateur devrait s'ouvrir vous montrant votre projet en exécution ! Wahou !

2.3 Ajouter un layer (couche)

OK, maintenant nous voulons ajouter plus d'objets. Cependant nous allons accidentellement continuer de sélectionner le *Tiled Background* à moins que nous ne le verrouillions (*lock*), le rendant ainsi non sélectionnable. Utilisons le système de *layers* pour ce faire.

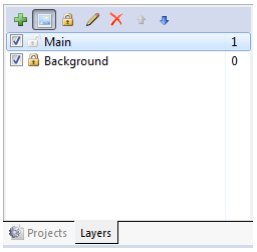
Les *layouts* peuvent être composés de multiples *layers* (couches) que nous pouvons utiliser afin de grouper les objets. Imaginez les *layers* comme des plaques de verre empilées les unes sur les autres avec les objets peints sur chaque plaque. Cela vous permet de réorganiser facilement quels objets apparaissent au-dessus des autres. Les *layers* peuvent aussi être cachés (invisibles), verrouillés, avoir des effets de parallaxe et bien plus. Par exemple, pour ce jeu, nous voulons que tout apparaisse par-dessus le *Tiled Background* donc nous pouvons ajouter une nouvelle couche pour nos autres objets.

Pour gérer les *layers*, cliquez sur l'onglet *layers* qui se trouve généralement à côté de la barre des projets :



Vous devriez voir Layer 0 dans la liste (Construct 2 compte à partir de zéro, car cela fonctionne mieux en programmation). Cliquez sur l'icône crayon et **rename** (renommer) le *layer* en *Background* (arrière-plan), car il s'agit de notre *layer* d'arrière-plan. Cliquez sur l'icône verte *add* (ajouter) pour ajouter un nouveau *layer* à la liste pour vos autres objets. Appelez ce nouveau layer *Main* (principal). Enfin, si vous cliquez sur l'icône « cadenas » à côté de *Background*, le *layer* se retrouvera verrouillé. Cela signifie que vous ne pourrez plus sélectionner les objets qui se trouvent sur ce *layer*. C'est assez pratique pour notre *Tiled Background*, qui peut être accidentellement sélectionné alors que vous n'avez plus besoin d'y toucher. Cependant, si vous avez besoin d'effectuer des changements, vous pouvez toujours cliquer de nouveau sur le cadenas pour déverrouiller le *layer*.

Les cases à cocher vous permettent aussi de cacher les *layers* dans l'éditeur, mais nous n'avons pas besoin de cela pour l'instant. Votre barre de *layers* devrait ressembler à ceci maintenant :



Maintenant, **assurez-vous que le layer 'Main' est sélectionné dans la barre des layers**. C'est très important, le *layer* sélectionné est le *layer* actif. Tous les nouveaux objets seront insérés dans le *layer* actif, donc s'il n'est pas sélectionné, nous insérerons accidentellement des objets dans le mauvais *layer*. Le *layer* actif est indiqué dans la barre d'état et apparaît aussi en tant qu'infobulle quand vous insérez un nouvel objet, gardez un ?il dessus.

2.4 Ajout des objets de contrôle (input)

Revenons au layout. Double-cliquez pour insérer un nouvel objet. Cette fois sélectionnez l'objet **Mouse** (souris) car vous allez avoir besoin de gérer les contrôles avec la souris. Faites de même pour l'objet **Keyboard** (clavier).

Ces objets n'ont pas besoin d'être positionnés dans le *layout*. Ils sont cachés et automatiquement accessibles à travers tout le projet. Désormais, tous les *layouts* de votre projet peuvent utiliser les contrôles de la souris et du clavier.

2.5 Les objets du jeu

Il est temps d'insérer les objets du jeu à proprement parler ! Voici des textures pour cela, sauvegardez-les sur votre disque dur comme précédemment.

Joueur :	
Monstre :	
Balle :	
et Explosion :	

Pour chacun de ces objets nous allons utiliser un objet *sprite*. Cet objet affiche la texture et peut être déplacé, tourné et redimensionné. En général les jeux vidéo affichent principalement des *sprites*. Insérons **chacun** des quatre objets précédents en tant que nouveau *sprite*. Le processus est similaire à l'insertion du *Tiled Background* :

1. **Double-cliquez** pour insérer un nouvel objet ;
2. **Double-cliquez** sur l'objet 'Sprite' ;
3. Quand le curseur de la souris affiche une croix, cliquez dans le *layout*. L'infobulle devrait être 'Main'. (souvenez-vous, il s'agit du *layer* actif) ;
4. L'éditeur de texture apparaît. Cliquez sur l'icône « ouvrir » (le petit dossier) et **chargez l'une des quatre textures précédentes** ;
5. **Fermez** l'éditeur de texture, sauvegardant ainsi vos changements. Vous devriez maintenant voir les objets dans votre *layout* !

Note : une autre manière rapide d'insérer des objets *sprite* est de glisser/déposer le fichier image depuis l'explorateur Windows dans l'espace du *layout*. Construct 2 créera automatiquement l'objet Sprite avec la texture pour vous. Assurez-vous de glisser chaque image une par une cependant, si vous glissez les quatre à la fois, Construct 2 créera un seul objet Sprite contenant quatre pas d'animation.

Déplacez les Sprites *balle* et *explosion* en dehors des limites du *layout* (hors de l'espace blanc), vous ne voulez pas voir ces objets au démarrage de votre jeu.

Ces objets seront nommés *Sprite*, *Sprite2*, *Sprite3* et *Sprite4*. Ce n'est pas très pratique et peut vite devenir très difficile à gérer. Renommez les objets en tant que *Player* (joueur), *Monster* (monstre), *Bullet* (balle) et *Explosion* (Note de traduction : je préfère garder les noms anglais ici, car les exemples d'évènements plus tard dans le tutoriel auront ces mêmes noms).

Pour renommer, sélectionnez un objet et modifiez sa propriété **Name** (nom) dans la barre des propriétés :

3 Ajouter des behaviors (comportements)

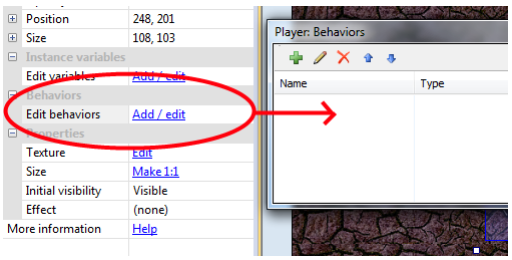
Les *behaviors* (comportements) sont des fonctionnalités fournies avec Construct 2. Par exemple, vous pouvez ajouter le *behavior* Platform à un objet, le *behavior* Solid à votre sol et instantanément vous pourrez déplacer et faire sauter votre personnage comme dans un jeu de plate-forme. Vous pouvez faire la même chose à l'aide des *events* (évènements) mais cela prend plus de temps et n'a pas toujours d'intérêt si un *behavior* permet déjà de le faire de manière appropriée. Jetons un ?il aux *behaviors* que nous pouvons utiliser. Entre autres Construct 2 dispose de ces *behaviors* :

- **8 Direction movement** (mouvement huit directions). Cela vous permet de déplacer un objet à l'aide des flèches du clavier et vous permettra de faire les mouvements du joueur (objet *Player*) ;
- **Bullet movement** (mouvement projectile). Ce comportement déplace un objet en ligne droite selon son angle actuel. Il fonctionnera très bien pour les balles (objet *Bullet*) que tirera le joueur. Malgré son nom il fonctionnera aussi très bien pour déplacer les monstres puisque tout ce que le comportement fait est de déplacer l'objet en avant à une vitesse constante ;
- **Scroll to** (défiler jusqu'à). Ceci fait que l'écran suit la position de l'objet alors que celui-ci se déplace (le principe du *scrolling*). Ceci sera utile pour le joueur ;
- **Bound to layout** (limité au *layout*). Cela évitera que l'objet sorte des limites du *layout*. Cela sera aussi utile pour le joueur afin qu'il ne sorte pas de l'écran ;
- **Destroy outside layout** (détruire hors du *layout*). Au lieu d'empêcher un objet de quitter l'écran, ce comportement détruit l'objet une fois que celui-ci sort des limites du *layout*. Cela vous servira pour les balles. Sans ce comportement, les balles quitteraient l'écran et continueraient à voler à l'infini, continueraient d'utiliser quelques ressources mémoires et processeur. Ce comportement permettra de détruire directement ces objets une fois qu'ils ne seront plus utiles dans la logique du jeu, qu'elles auront quitté l'écran ;
- **Fade** (disparition graduelle). Ce comportement fait disparaître graduellement l'objet. Il sera utilisé pour les explosions.

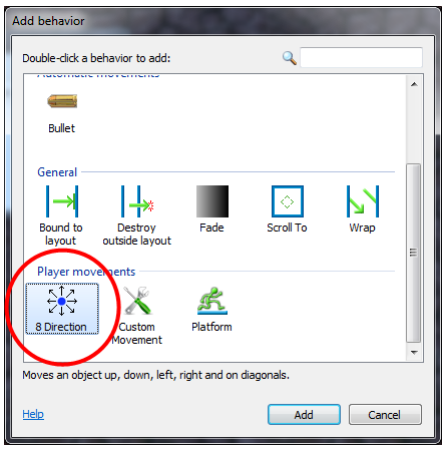
Ajoutons les comportements aux objets qui en ont besoin.

3.1 Comment ajouter un behavior

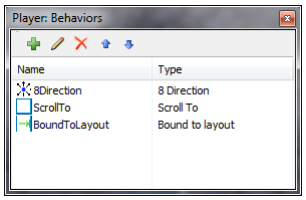
Ajoutons le *behavior* **8 direction movement** au joueur (objet *Player*). Cliquez sur « Player » pour le sélectionner. Dans la barre des propriétés, notez la catégorie *Behaviors*. Cliquez sur *Add/Edit* (ajouter/éditer) dans cette catégorie. La fenêtre des *behaviors* (behaviors dialog) pour l'objet *Player* va s'ouvrir.



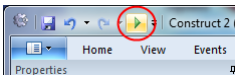
Cliquez sur l'icône verte *add behavior* (ajouter un *behavior*) dans la fenêtre. Double-cliquez le *behavior* **8 direction movement** pour l'ajouter.



Faites la même chose et cette fois ajoutez le *behavior Scroll To* afin de faire que l'écran suive l'objet « Player », ainsi que le *behavior* **Bound to layout** pour limiter sa position aux limites du *layout*. La fenêtre des *behaviors* devrait ressembler à ceci maintenant :



Fermez la fenêtre des *behaviors* et cliquez sur *Run* (exécuter) pour essayer le jeu !



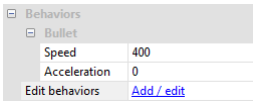
Espérons que vous avez un navigateur compatible HTML 5 installé. Autrement soyez aussi sûrs d'avoir la dernière version de FireFox, Chrome ou encore Internet Explorer 9, si vous êtes sous Windows Vista et supérieur. Une fois que votre jeu est en cours d'exécution, notez que vous pouvez déjà déplacer le joueur avec les flèches du clavier et que l'écran le suit ! Remarquez aussi que le joueur ne sort pas des limites du *layout* grâce au *behavior* « Bound to layout ». C'est là l'utilité des *behaviors* : rapidement ajouter des fonctionnalités communes. Nous allons bientôt utiliser le système d'évènements pour ajouter des fonctionnalités personnalisées.

3.2 Ajouter les autres behaviors

Nous pouvons ajouter des comportements aux autres objets par la même méthode. Sélectionnez l'objet, cliquez sur *Add/Edit* pour ouvrir la fenêtre des *behaviors* et ajoutez certains *behaviors*. Ajoutez donc ces autres *behaviors* :

- ajoutez les *behaviors* **Bullet movement** et **Destroy outside layout** à l'objet **Bullet** (balle) (pas de surprises ici) ;
- ajoutez le *behavior* **Bullet movement** à l'objet **Monster** (monstre), car il ne fait que bouger en avant aussi ;
- ajoutez le *behavior* **Fade** à l'objet **Explosion** (afin qu'il disparaisse progressivement après être apparu). Par défaut le *behavior* Fade détruit l'objet une fois la disparition effectuée, ainsi vous n'aurez pas à vous soucier des objets *Explosions* invisibles utilisant la mémoire et des performances du jeu.

Si vous exécutez le jeu maintenant, vous pouvez remarquer que la seule différence est que les monstres se déplacent plutôt rapidement. Ralentissez-les un peu pour un rythme plus appréciable. Sélectionnez l'objet **Monster** (monstre). Notez que puisque vous avez ajouté un *behavior* de nouvelles propriétés sont disponibles dans la barre des propriétés :



Cela vous permet d'adapter et modifier le fonctionnement des *behaviors*. Changez lavitesse (*speed*) de **400** à **80** (il s'agit du déplacement en pixels par seconde).

De la même manière changez la vitesse (*speed*) de l'objet **Bullet** (objet balle) à 600 et la propriété *Fade out time* du *behavior Fade* de l'objet **Explosion** à **0.5** (une demi-seconde).

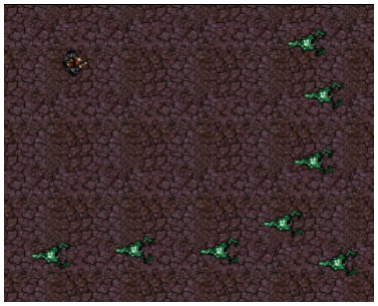
3.3 Créez des monstres

En maintenant la touche Ctrl, cliquez sur l'objet *Monster* (Monstre) et déplacez-le. Notez que cette opération crée une nouvelle instance de l'objet. C'est un autre objet du type d'objet Monster.

Les types d'objet sont essentiellement des 'classes' d'objets. Dans le système d'évènements vous avez principalement à faire à des types d'objet. Par exemple, vous pouvez créer un évènement qui dit « Bullet collides with Monster » (Balle est en collision avec un monstre). Cela veut en fait dire « N'importe quelle instance du type d'objet Bullet (balle) rentre en collision avec n'importe quelle instance du type d'objet Monster (monstre) ». Ceci permet de ne pas avoir à créer un évènement pour chaque (instance) monstre. Avec des Sprites toutes les instances du type d'objet partagent les même textures. Cela est très efficace quand des joueurs jouent à ce jeu en ligne, au lieu d'avoir à télécharger huit textures de monstres pour huit instances, ils n'ont besoin de télécharger qu'une fois la texture et Construct 2 l'applique huit fois.

Nous couvrirons plus en détails la différence entre *type d'objet* et *instances* plus tard. Pour l'instant un bon exemple est de penser que différents types d'ennemis sont des types d'objets différents et que les ennemis d'un même type sont des instances de ce type d'objet.

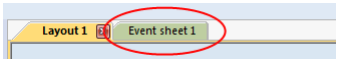
En utilisant la touche Ctrl et déplacer, **créez sept ou huit monstres**. Ne les placez pas trop près du joueur ou ils pourraient mourir rapidement ! Vous pouvez dézoomer en maintenant Ctrl et la molette de la souris, si ça aide et dispersez-les à travers l'intégralité du *layout*. Vous devriez obtenir quelque chose qui ressemble à cela :



Maintenant il est temps d'ajouter quelques fonctionnalités personnalisées à l'aide de la méthode de programmation de Construct 2 : le *système d'évènements* (event system).

4 Évènements

D'abord, cliquez sur l'onglet *Event sheet 1* (feuille d'évènements 1) en haut pour passer sur l'éditeur d'*event sheet*. Une liste d'évènements est appelée *Event sheet* et vous pouvez avoir différentes *event sheets* pour différentes parties de votre jeu ou juste pour une meilleure organisation. Les *events sheets* peuvent aussi « include » (inclure) d'autres *event sheets*, vous permettant de réutiliser des évènements dans plusieurs niveaux par exemple, mais nous n'en aurons pas besoin tout de suite.



4.1 À propos des évènements

Comme le texte dans la feuille d'évènements vide l'indique Construct 2 exécute tout dans cette feuille une fois par tick. La plupart des écrans mettent à jour leur affichage 60 fois par seconde donc, Construct 2 va essayer d'avoir le même débit afin d'assurer un rendu adapté. Cela signifie que l'*event sheet* sera généralement exécutée 60 fois par seconde, redessinant l'écran à chaque fois. C'est cela un tick, une unité de « exécute les évènements et dessine l'écran ».

Les évènements s'exécutent de haut en bas, donc les évènements placés en haut de la feuille d'évènements seront exécutés en premier.

4.2 Conditions, actions et sub-events (sous-évènements)

Les évènements sont constitués de **conditions** qui testent si certains critères sont remplis (par exemple : « est-ce que la touche espace est enfoncée ? »). Si toutes les conditions sont remplies, les **actions** de l'évènement sont toutes exécutées (par exemple : « créer un objet balle »). Une fois que les actions ont été exécutées, tout **sub-event** (sous-évènement) est également exécuté (ces *sub-events* peuvent tester d'autres conditions, exécuter d'autres actions, ainsi que d'autres *sub-events* et ainsi de suite). En utilisant ce système, nous pouvons créer des fonctionnalités complexes pour nos jeux et applications. Cependant pour ce tutoriel nous n'aurons pas besoin de sub-events.

Revoyons ceci une fois de plus. En résumé un évènement s'exécute ainsi de base :

Est-ce que toutes les conditions sont remplies ?

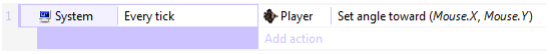
—> **Oui** : exécute toutes les actions de l'évènement.

—> **Non** : aller à l'évènement suivant (en sautant les *sub-events* de l'évènement actuel).

Il s'agit d'une simplification du déroulement. Construct 2 fournit beaucoup de fonctionnalités d'évènements qui couvrent et permettent pas mal de choses dont vous pouvez avoir besoin. Cependant, pour l'instant, c'est une bonne manière de voir les choses.

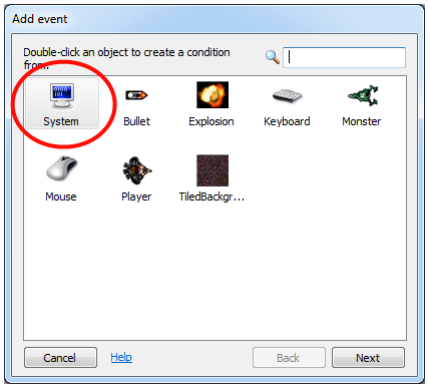
4.3 Votre premier évènement

Nous voulons faire en sorte que l'objet *Player* (joueur) regarde toujours en direction du curseur de la souris. Cela ressemblera à ceci une fois achevé :



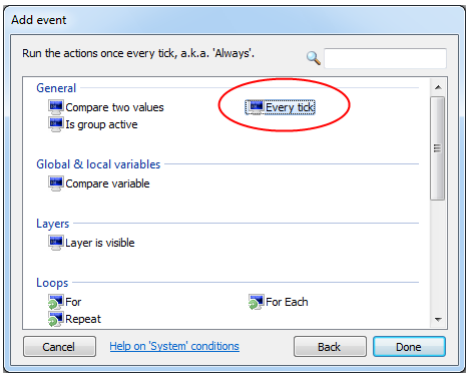
Souvenez-vous, un tick s'exécute à chaque fois que l'écran est redessiné. Donc si vous faites en sorte que le joueur pointe dans la direction du curseur à chaque tick, il apparaîtra regardant vers le curseur durant toute l'exécution.

Commençons par cet évènement. Double-cliquez dans la feuille d'évènements. Cela va afficher une fenêtre nous permettant d'ajouter une **condition** pour le nouvel évènement.



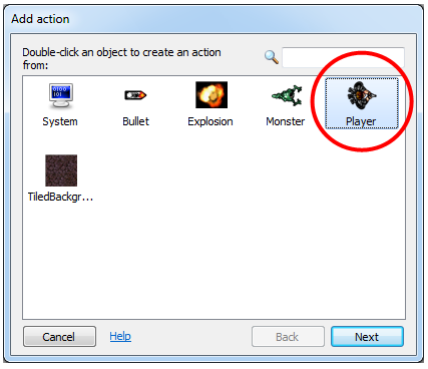
Différents objets ont différentes conditions et actions en fonction de ce qu'ils peuvent faire. Il y a aussi un **objet System** (système) qui contient les fonctionnalités « standards » intégrées à Construct 2. **Double-cliquez** sur l'objet *system* comme indiqué.

La fenêtre affichera alors une liste de toutes les conditions de l'objet *System* :

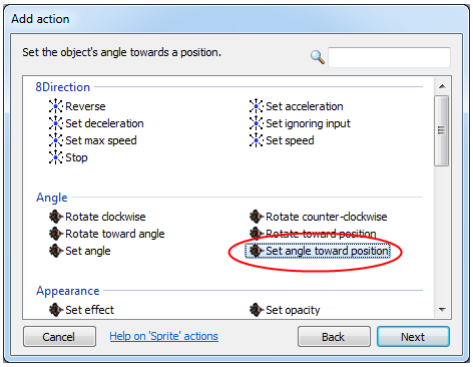


Faites un double-clic sur la condition *Every tick* (chaque tick) pour l'insérer. La fenêtre se fermera et l'évènement sera créé sans action. Votre feuille devrait désormais ressembler à ceci :

Maintenant nous voulons ajouter une action pour faire en sorte que l'objet *Player* (joueur) regarde vers la souris. Cliquez sur le lien *Add action* (ajouter action) à droite de l'évènement (assurez-vous de cliquer sur le lien « Add action », pas « Add event » en dessous qui créerait un tout nouveau niveau d'évènement). La fenêtre « Add action » s'affichera :



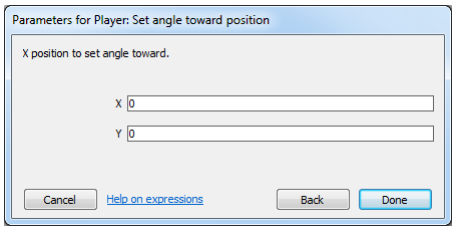
Comme pour l'ajout d'un évènement, nous avons la même liste d'objets dans laquelle on peut choisir, mais cette fois-ci, pour l'ajout d'une action. Essayez de ne pas vous mélanger les pinceaux entre l'ajout de condition et l'ajout d'action ! Comme montré, **double-cliquez** sur l'objet « Player », car c'est cet objet que nous souhaitons faire pointer vers le curseur. La liste des actions disponibles pour l'objet « Player » apparaît :



Remarquez comme le *behavior 8-direction movement* de l'objet *Player* a ses propres actions. Nous n'avons pas besoin de nous en soucier pour l'instant.

Plutôt que de définir l'angle du *Player* d'un certain nombre de degrés, il est plus pratique d'utiliser l'action **Set angle towards position** (définir l'angle vers une position). Cela calculera automatiquement l'angle entre l'objet « Player » et les coordonnées X et Y soumises et l'appliquera à l'objet. **Double-cliquez** sur l'action **Set angle towards position**.

Construct 2 a besoin de connaître les coordonnées X et Y vers lesquelles le joueur doit pointer :



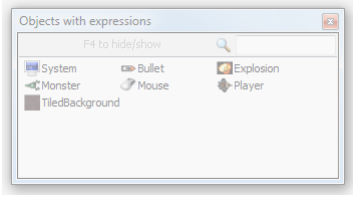
Ceux-ci sont appelés **paramètres** de l'action. Les conditions peuvent avoir des paramètres aussi, mais *Every tick* n'en a besoin d'aucun.

Vous voulez définir l'angle vers la position de la souris. L'objet « Mouse » (souris) vous fournit un moyen de le faire. Entrez **Mouse.X** pour **X** et **Mouse.Y** pour **Y**. Ce sont des expressions. Elles sont telles des formules mathématiques calculées. Par exemple vous pourriez aussi écrire Mouse.X + 100 ou sin(Mouse.Y) (bien que ces exemples précis ne soient pas particulièrement utiles !). De cette manière vous pouvez utiliser n'importe quelles données issues d'un objet, ou n'importe quel calcul pour élaborer les paramètres dans des actions et des conditions. C'est très puissant et une sorte d'origine secrète de la flexibilité de Construct 2.

Avez-vous rencontré une erreur qui disait « Mouse is not an object name » (Mouse n'est pas un nom d'objet) ? Assurez-vous d'avoir ajouté l'objet *Mouse* (souris) ! Retournez au paragraphe « Ajout des objets de contrôle (input) ».

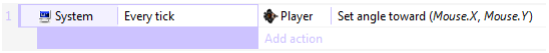
Vous pouvez vous demander comment se rappeler toutes les expressions que vous pourriez écrire/utiliser. Heureusement il y a un « objet panel » (panneau d'objet) que vous devriez voir flotter en des-

sous de la fenêtre. Par défaut le panneau est semi-transparent afin de ne pas vous distraire.



Placez votre curseur dessus, ou cliquez dessus et le panneau deviendra complètement visible. Cela sert comme dictionnaire de toutes les expressions que vous pouvez utiliser, avec une description pour vous aider à les mémoriser. Si vous double-cliquez sur un objet vous verrez la liste de ses expressions disponibles. Si vous double-cliquez sur une expression elle sera insérée, vous évitant d'avoir à la taper manuellement.

Quoiqu'il en soit cliquez sur **Done** (Fini) dans la fenêtre de paramètres. L'action est ajoutée ! Comme vu précédemment l'évènement devrait ressembler à cela :



Et voilà, votre premier évènement ! Essayez d'exécuter le jeu et le joueur devrait pouvoir se déplacer tout en faisant toujours face au curseur de la souris. Voilà votre premier bout de fonctionnalité personnalisée.

5 Ajouter des fonctionnalités au jeu

Si chaque évènement doit être décrit avec autant de détails que précédemment, cela risque d'être un très long tutoriel. Accordez-vous pour rendre la description un peu plus brève pour les prochains évènements. Souvenez-vous, les étapes pour ajouter une condition ou une action sont :

1. Double-cliquez pour insérer un nouvel évènement, ou cliquez sur le lien *Add action* pour ajouter une action ;
2. Double-cliquez sur l'objet dans lequel la condition/action se trouve ;
3. Double-cliquez sur la condition/action que vous voulez ;
4. Entrez les paramètres au besoin.

À partir de maintenant les évènements seront décrits comme l'objet, suivi de condition/action, suivi des paramètres. Par exemple, l'évènement que nous venons d'insérer pourrait s'écrire :

Add condition *System -> Every tick*

Add action *Player -> Set angle towards position ? X : Mouse.X, Y : Mouse.Y*

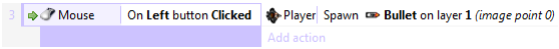
5.1 Faire en sorte que le Player (joueur) tire

Quand le joueur clique sur l'objet *Player* celui-ci devrait tirer une balle. Cela peut être fait à l'aide de l'action *Spawn an object* de l'objet *Player* (joueur), qui créé une nouvelle instance d'un objet à la même position et angle qu'un objet référent. Le *behavior Bullet movement* que nous avons ajouté plus tôt le fera ensuite voler/avancer vers l'avant. Faites l'évènement suivant :

Condition : *Mouse -> On click ->* Left clicked (par défaut)

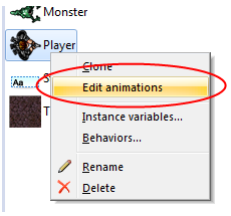
Action : *Player -> Spawn another object ->* Pour *Object*, choisissez l'objet *Bullet* (balle). Pour Layer, marquez **1** (le layer « Main » est le layer 1 - souvenez-vous Construct 2 compte à partir de zéro). Laissez *Image point* à 0.

Votre évènement devrait ressembler à ça :

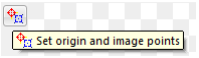


Si vous exécutez le jeu vous remarquerez que les balles partent du milieu du corps du joueur plutôt que du bout de son arme. Corrigeons cela en ajoutant un *image point* au bout de l'arme (un « image point » est juste une position dans l'image depuis laquelle vous pouvez créer/placer des objets.).

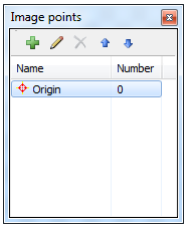
Faites un clic droit sur l'objet « Player » (joueur) dans la barre de projet ou d'objet et sélectionnez **Edit animations** (éditer les animations).



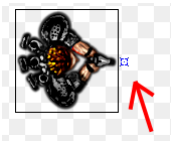
L'éditeur d'images pour le joueur apparaît de nouveau. Cliquez sur l'outil « origin and image points » :



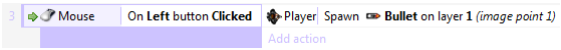
. . . et la fenêtre d'« image points » s'ouvre :



Notez que le point d'origine de l'objet apparaît en tant que point rouge. C'est le « hotspot » ou « point de pivot » de votre objet. Si vous tournez l'objet, il tourne autour de son point d'origine. Nous désirons ajouter un autre « image point » pour représenter l'arme donc cliquez sur le bouton vert *add* (ajouter). Un point bleu apparaît, c'est notre nouvel « image point ». Faites un clic gauche sur le bout de l'arme du joueur afin de placer l'« image point » à cet endroit :



Fermez l'éditeur d'image. Double-cliquez sur l'action *Spawn an object* que nous avons ajouté tout à l'heure et changez *Image* point à 1 (le point d'origine sera toujours le premier point et rappelez-vous que Construct 2 compte en partant de zéro). Cet évènement devrait maintenant rassembler à ceci :



et notez bien que cela dit *Image point 1* maintenant.

Lancez le jeu. Les balles partent maintenant du bout de votre arme ! Cependant, les balles n'ont aucun effet. Toutefois, vous allez vous rendre compte, lorsque vous aurez pratiqué le système d'évènements, à quel point l'ajout de fonctionnalités est rapide.

Maintenant faisons en sorte que les balles (objet *Bullet*) tuent les monstres. Ajoutons l'évènement suivant :

Condition : *Bullet* -> *On collision with another object* -> Sélectionne l'instance *Monster* (monstre).

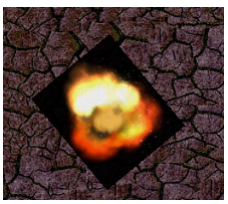
Action : *Monster* -> *Destroy*

Action : *Bullet* -> *Spawn another object* -> *Explosion*, layer 1

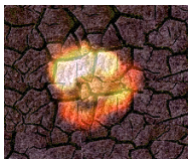
Action : *Bullet* -> *Destroy*

5.2 L'effet d'explosion

Exécutez le jeu et essayez de tirer sur un monstre. Oups, l'explosion a une grosse bordure noire !



Vous aviez peut-être prévu que cela ressemblerait à l'explosion ci-dessus puis vous vous êtes demandé si votre jeu allait vraiment ressembler à cela ! Ne vous inquiétez pas, ce ne sera pas le cas. **Cliquez sur l'objet Explosion**, soit dans la barre d'objets en bas à droite, ou dans la barre de projet (qui est en onglet avec la barre de *layers*). Les propriétés de l'objet apparaissent dans la barre des propriétés à gauche. En bas, sélectionnez la valeur **Addidtive** de la propriété **Blend mode**. Maintenant essayez de nouveau le jeu.



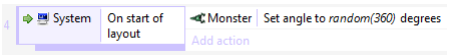
Pourquoi cela fonctionne-t-il ? Sans entrer dans les détails techniques, les images ordinaires sont *copiées par-dessus* l'écran. Avec l'effet « additive », chaque pixel est *ajouté* (comme dans une addition) avec le pixel d'arrière-plan sous lui. Noir est un pixel de valeur 0, donc rien ne s'ajoute, vous ne voyez pas l'arrière-plan noir. Les couleurs plus claires ajoutent plus et donc apparaissent plus visiblement. Ceci est super pour les explosions et les effets de lumière.

5.3 Rendre les monstres un peu plus malins

Pour l'instant, les monstres se « promènent » vers l'extérieur du *layout* sur la droite. Rendons-les un peu plus intéressants. Tout d'abord, donnons leur un angle aléatoire au départ.

Condition : *System* -> *On start of Layout*

Action : *Monster* -> *Set angle* -> random(360)



Ils erreront encore pour toujours lorsqu'ils quittent le *layout*. Gardons-les à l'intérieur. Ce que nous allons faire est de les faire pointer en direction du joueur (objet Player) lorsqu'ils quittent l'espace du layout. Cela permet deux choses : ils restent toujours dans le *layout* et si le joueur ne se déplace pas les monstres iront directement sur lui !

Condition : *Monster* -> *Is outside layout*

Action : *Monster* -> *Set angle toward position* -> Pour X, **Player.X** - Pour Y, **Player.Y**.

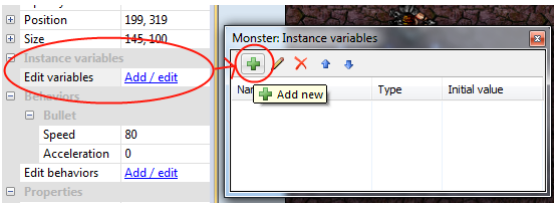
Exécutez le jeu. Si vous attendez un peu vous allez remarquer que les monstres restent dans l'espace du *layout* et qu'ils vont dans toutes sortes de directions. C'est à peine une IA (intelligence artificielle) mais cela sera suffisant.

Maintenant, supposons que nous voulions tirer sur un monstre cinq fois avant qu'il ne meure au lieu d'instantanément comme à l'heure actuelle. Comment réalisons-nous cela ? Si nous utilisons un seul compteur « Health » (santé) alors lorsque nous aurons touché un monstre cinq fois *tous* les monstres vont disparaître. À la place, nous avons besoin que chaque monstre se « souvienne » de *sa propre santé*. Nous pouvons y parvenir à l'aide des **variables d'instance** (instance variables).

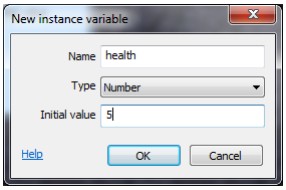
6 Variables d'instance (Instance variables)

Les variables d'instance permettent à chaque monstre de stocker la valeur de sa propre santé. Une variable est juste une valeur qui change (ou *varie*) et elles sont stockées séparément pour chaque instance d'où le nom *variable d'instance*.

Ajoutez une variable d'instance *health* (santé) à notre objet *Monster* (monstre). Cliquez sur l'objet dans la barre de projet ou d'objets. Vous pouvez aussi revenir sur la vue de *layout* et cliquer sur l'objet « Monster » directement. Cela montrera les propriétés de l'objet dans la barre des propriétés. Cliquez sur **Add/edit** (Ajouter/éditer) dans la section **Edit variables** (Éditer les variables).

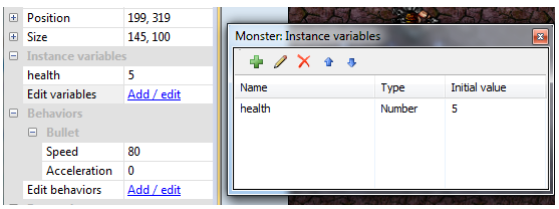


La fenêtre des variables d'instance s'affiche. Elle ressemble à la fenêtre des *Behaviors* que vous avez vue précédemment, mais vous permet d'ajouter et de modifier les variables d'instance de l'objet. Cliquez sur le bouton vert **Add** (ajouter) pour en ajouter une nouvelle.



Dans la boîte de dialogue qui apparaît, tapez **health** pour le nom (name), laissez Type en tant que **Number** (nombre/numérique) et comme *Initial value* (valeur initiale) entrez **5** (comme illustré). Cela donnera cinq points de vie à tous les monstres dès le départ. Quand ils seront touchés la valeur de « health » sera diminuée de 1 et quand « health » tombe à zéro l'objet sera détruit.

Une fois terminé, cliquez sur OK. Remarquez que la variable apparaît maintenant dans la fenêtre de variables d'instance ainsi que dans les propriétés du monstre (vous pouvez rapidement modifier la valeur de départ dans la barre des propriétés, mais pour ajouter ou supprimer une variable d'instance vous devrez cliquer sur le lien *Add/Edit* (Ajouter/Editer).).

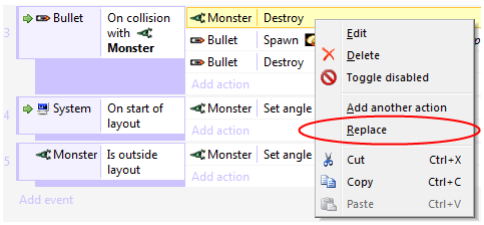


6.1 Modifier les évènements

Retournez dans la feuille d'évènements (*Event sheet*). Pour l'instant nous détruisons les monstres dès qu'ils sont touchés par une balle. Modifions cela afin de soustraire un point de vie.

Trouvez l'évènement : *Bullet - on collision with Monster*.

Notez que nous avons une action « destroy monster ». Remplacez-la par une action « subtract 1 from health ». Clic droit sur l'action « destroy monster » et cliquez sur **Replace** (remplacer).



La même fenêtre que si vous étiez en train d'ajouter une action s'affiche, mais cette fois elle remplacera l'action que vous avez sélectionnée à la place. Choisissez *Monster => Substract from* (dans la catégorie *Instance variables*) -> la variable d'instance « health » et entrez **1** en tant que Value (valeur). Cliquez sur **Done** (fini). L'action devrait maintenant apparaître comme suit :

Maintenant quand nous touchons un monstre avec une balle, il perd un point de vie et la balle explose mais nous n'avons pas encore fait d'évènement pour tuer le monstre quand sa vie est à 0. Ajoutons un nouvel évènement :

Condition : *Monster -> Compare instance variable* -> Health, *Less or equal*, 0

Action : *Monster -> Spawn another object* -> Explosion, layer 1

Action : *Monster -> Destroy*

Pourquoi « less or equal 0 » (moins ou égal à 0) plutôt que « equals 0 » (égal à 0) ? Supposons que vous ajoutiez une arme plus puissante qui soustrait 2 de *health* (santé). Lorsque vous tirez sur un monstre, sa santé ira de **5**, **3**, **1**, **-1**, **-3** ?

Remarquez qu'à aucun moment *health* ne sera directement *égal à* 0, donc le monstre ne mourrait jamais !

Ainsi c'est une pratique correcte d'utiliser « less or equal » (moins ou égal) pour tester si quelque chose n'a plus de vie (health).

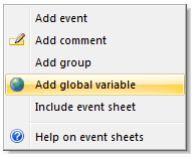
Exécutez le jeu. Vous devez désormais toucher les monstres cinq fois pour les tuer !

7 Garder le score

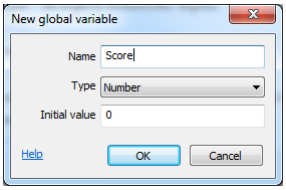
Maintenant, ajoutons un score pour que le joueur sache qu'il a bien joué. Nous aurons besoin d'une variable de plus pour ceci. Vous pourriez penser « mettons le score en tant que variable d'instance de l'objet 'Player' ». Ce n'est pas mal a priori, mais souvenez-vous que la valeur sera stockée « dans » l'objet. S'il n'y a pas d'instance de l'objet, il n'y a pas de variables d'instance non plus ! Donc si vous détruisez le joueur, vous ne pouvez plus savoir quel était le score car la valeur a disparu avec l'instance.

À la place, utilisez une **global variable** (variable globale). Comme une variable d'instance, une variable globale (ou juste « globale ») peut stocker du texte ou un nombre. Chaque variable peut stocker un nombre ou un seul morceau de texte. Les variables globales sont aussi disponibles à travers tous les *layouts*, utile si vous voulez ajouter d'autres niveaux.

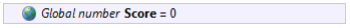
Faites un clic droit dans l'espace en bas de l'*event sheet* (feuille d'évènements) et sélectionnez *Add global variable* (ajouter une variable globale).



Entrez **Score** en tant que nom (*name*). Les autres champs sont OK, cela fera commencer le nombre (*number*) à 0.



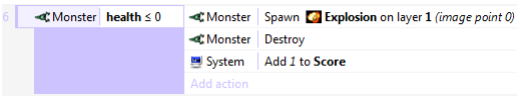
Maintenant la variable globale apparaît en tant qu'une ligne dans la feuille d'évènements. Elle est placée dans cette *event sheet* mais est accessible depuis n'importe quelle autre *event sheet* depuis n'importe quel autre *layout*.



Il y a aussi des *local variables* (variables locales) qui ont une plus faible « portée » et ne peuvent être accédées que par des évènements « proches ». Pour l'instant nous n'avons pas besoin de nous soucier de celles-ci.

Donnez au joueur un point à chaque fois qu'il tue un monstre. Dans votre évènement « Monster : health less or equal 0 » (la variable d'instance « health » (santé) de l'objet « Monster » (monstre) est inférieure ou égale à 0 quand un monstre meurt, donc)), cliquez sur Add action (ajouter une action) et sélectionnez *System* -> *Add to* (ajouter dans la catégorie « Global & local variables ») -> **Score**, value (valeur) **1**.

Maintenant l'évènement devrait ressembler à ceci :



Désormais le joueur a un score qui augmente de 1 pour tout monstre tué, mais il ne peut pas le voir ! Affichons le score à l'aide de l'objet « text » (texte).

8 Créer un affichage tête haute (heads-up display (HUD))

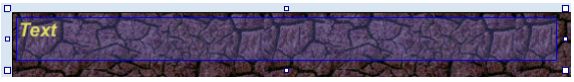
Un affichage tête haute (aussi appelé HUD en anglais) est l'interface qui montre la vie (*health*) du joueur, son score et d'autres informations en cours de jeu. Créons un HUD très simple à l'aide de l'objet « text ».

Le HUD reste toujours à la même place à l'écran. Si nous avons quelques objets d'interface, nous ne voulons pas qu'ils « scrollent » à mesure que le joueur (*Player*) se déplace, ils devraient plutôt rester à l'écran en permanence. Par défaut les *layers* « glissent » (*scroll*). Pour les garder à l'écran, nous pouvons utiliser le réglage **Parallax** des *layers*. La parallaxe permet à différents *layers* de « scroller » à différentes vitesses les uns par rapport aux autres dans un effet de pseudo 3D. Si nous mettons la parallaxe d'un *layer* à 0, le *layer* ne glissera pas du tout, idéal pour un HUD.

Retournez à la barre de couches (*layers*) que nous avons utilisé précédemment. Ajoutez un nouveau *layer* nommé *HUD*. Assurez-vous qu'il soit au niveau supérieur (en premier dans la liste) et qu'il soit sélectionné (souvenez-vous cela active le *layer*). La barre des propriétés devrait afficher ses propriétés. Fixez la valeur de la propriété **Parallax** à **0**, **0** (**0** sur l'axe des X et des Y).

Double-cliquez dans un espace du *layout* pour insérer un nouvel objet. Cette fois-ci sélectionnez l'objet **Text** (texte). Placez-le dans le coin supérieur gauche de votre *layout*. Cela sera difficile à voir si le fond est noir, donc dans la barre des propriétés rendez l'objet en gras (*bold*), italique (*italic*), de couleur jaune et choisissez une taille un peu supérieure pour

la police. Redimensionnez l'objet de manière assez large pour qu'il puisse afficher tout le texte. Cela devrait ressembler à quelque chose comme ça :



Retournez à la feuille d'évènements. Faites en sorte de mettre à jour le contenu de l'objet « Text » avec le score du joueur. Dans l'évènement **Every tick** (tous les ticks) que nous avons ajouté précédemment, ajoutez une action *Text* ? *Set text* (définir le texte).

En utilisant l'opérateur **&** vous pouvez convertir un nombre en texte et l'adjoindre à un autre bout de texte. Pour la valeur « Text » entrez :

« Score : » & Score

La première partie (« Score : ») veut dire que le texte affiché commencera toujours avec les mots Score :. La seconde partie (Score) est la valeur de la variable globale (*global variable*) « Score ». Le **&** « joint » les deux parties en une seule et même ligne de texte.

Exécutez le jeu et tirez sur des monstres. Votre score apparaît et reste à la même position à l'écran malgré vos déplacements !

9 Finitions

Vous avez presque fini. Ajoutons les dernières touches à notre jeu.

Tout d'abord, faites en sorte d'avoir des monstres qui « spawnent » (« naîssent ») de manière régulière autrement une fois que vous aurez tué tous les monstres créés dans l'éditeur, il n'y aura plus rien à faire. Vous allez créer un nouveau monstre toutes les trois secondes. Ajoutez un nouvel évènement :

Condition : *System* -> *Every X seconds* -> **3**

Action : *System* -> *Create object* -> **Monster**, layer **1**, **1400** (pour X), **random(1024)** (pour Y)

1400 est une coordonnée X juste à la droite du bord du *layout* et random(1024) est une coordonnée Y aléatoire qui appartient à la portée de la hauteur du *layout* (*height*).

Enfin, faites en sorte que les monstres tuent le joueur.

Condition : *Monster* -> *On collision with another object* -> **Player**

Action : *Player* -> *Destroy*

10 Conclusion

Félicitations, vous avez fait votre premier jeu en HTML 5 avec Construct 2 ! Si vous avez un serveur et que vous désirez montrer votre travail, cliquez sur **Export** (exporter) dans le menu *File* (fichier). Construct peut sauver tous les fichiers du projet dans un dossier de votre ordinateur que vous pouvez ensuite télécharger (*upload*) vers un serveur et/ou intégrer à une page web. Si vous n'avez pas de serveur disponible, vous pouvez partager votre jeu avec Dropbox : lien 72.

Vous avez appris des bases essentielles à propos de Construct 2 : insérer des objets, utiliser les *layers*, les *behaviors*, les évènements et plus encore. Espérons que ce tutoriel vous a bien préparé à apprendre plus de Construct 2 ! Essayez d'explorer ses fonctionnalités et voir ce qu'il peut faire pour vous.

10.1 Le truc fini

Essayez de télécharger le projet tutoriel complet : lien 73. J'ai (Ashley) ajouté des fonctionnalités en plus telles qu'un texte « Game over » et des monstres qui vont de plus en plus vite à mesure que le temps passe. Étant donné ce que vous savez maintenant, il ne devrait pas être trop difficile de deviner comment cela fonctionne. Il y a aussi beaucoup de commentaires (en anglais) qui décrivent le fonctionnement du jeu.

Bien joué ! Si vous avez le moindre problème ou si vous pensez que certaines parties du tutoriel pourraient être améliorées, laissez un commentaire ou un message dans les forums. Et montrez-nous de quoi vous êtes capables !

Enfin, si vous avez aimé ce tutoriel et pensez que l'une de vos connaissances pourraient aussi apprécier Construct 2, pourquoi ne pas lui envoyer le lien ? Cela ne peut faire de mal à personne.

10.2 Encore plus de lecture

Vous voulez ajouter de la musique et des effets sonores ? Allez lire Sounds & Music dans le manuel pour un aperçu rapide : lien 74.

Vous pouvez être intéressés par le guide du débutant alternatif basé sur les jeux de plate-forme : How to make a platform game (lien 75).

Si vous voulez en apprendre plus à propos de comment fonctionnent les évènements dans Construct 2 soyez sûrs de lire la section How Events Work dans le manuel : lien 76.

C'est hautement recommandé afin de vous permettre à créer rapidement vos propres projets !

Et pour encore plus d'informations, n'oubliez pas la documentation complète dans le manuel : lien 77 (en anglais à l'heure d'écriture).

Retrouvez l'article de ***Ashley Gullen*** *traduit par* ***Kyatric*** *en ligne : lien 78*

Commençons... avec des cercles

1 Introduction

La création de ses propres graphismes est une nécessité pour la plupart des développeurs indépendants. À cause des contraintes du budget ou dans la plupart des cas du manque de budget, de nombreux développeurs de jeux indépendants ne peuvent pas recruter un artiste ou acheter des graphismes.

Avec l'aide des logiciels libres comme Gimp, Inkscape, Truespace, DAZ Studio et Vue Pioneer (pour ne mentionner que certains d'entre eux) combinés aux notions de base de création de graphismes, presque tout le monde peut créer des résultats impressionnants et professionnels.

Je vais essayer de commencer avec des idées basiques et des exercices pour améliorer ces notions. Tous les exemples reposeront sur des logiciels libres. La manière de travailler sera identique si vous travaillez avec un autre logiciel comme Adobe Illustrator, Adobe Photoshop, CorelDraw ou autre. Je vais essayer de mentionner les différentes approches pour l'ensemble des logiciels. Avec la vaste variété de logiciels disponibles actuellement et l'infinité d'outils et de techniques de création de graphismes, il est impossible de tout couvrir, mais je vais essayer de garder mes exemples assez simples pour travailler avec les logiciels de votre choix.

Laissez-moi clarifier quelques idées reçues communes lorsqu'arrive l'étape de création de graphismes.

« J'ai besoin d'outils coûteux pour créer des graphismes vraiment professionnels. »

Non, vous n'en avez pas besoin ! Il y a de nombreux outils gratuits offrant une vraie alternative. Gimp est l'un des exemples les plus connus dans le royaume de la 2D ainsi que Blender pour la 3D. Pour un graphiste à temps complet, la mise à jour de vos outils afin de rejoindre « le standard industriel » est logique. Surtout pour faciliter les échanges et le partage lors de la coopération avec d'autres artistes utilisant des fichiers de format standard.

« Acheter un outil cher créera automatiquement des graphismes meilleurs. »

Non, cela ne le fera pas. C'est toujours l'artiste créant les graphismes qui les rend bien. Même avec les outils les plus simples comme le papier et le crayon, un bon artiste peut créer des pièces impressionnantes alors que les outils les plus sophistiqués nécessitent toujours un bon artiste pour créer quelque chose de spécial.

« Je ne peux pas faire de graphisme. Je ne sais même pas faire un bonhomme en bâtons. »

Si, vous pouvez. C'est là que les ordinateurs modernes entrent en scène et vous permettent de créer de bons graphismes sans avoir un diplôme en art.

« Mon jeu est bon tel quel. Je n'ai pas besoin de graphismes. »

Si, vous en avez besoin. Le marché du jeu indépendant s'étend de plus en plus et est plus attractif que dans le passé. Afin que votre jeu se démarque, il doit tout faire : avoir un bon gameplay aussi bien que des visuels consistants et attrayants, ainsi que des sons et des musiques qui correspondent.

1.1 Obstacles communs

1.1.a Trop d'espoir

L'un des principaux problèmes des développeurs indépendants est qu'ils attendent trop d'eux-mêmes. Pour les développeurs seuls ou les petits studios, il est presque impossible de faire un jeu AAA possédant la qualité des grands studios. Vous devez essayer et viser les étoiles. Réaliser des jeux consiste en cela. Faites de votre mieux et repoussez constamment vos limites en améliorant vos talents... mais pensez de façon réaliste et réajustez vos attentes par rapport à vos compétences et votre budget. Ce sera un grand pas en avant pour réaliser le meilleur jeu que vous puissiez faire.

1.1.b Définissez un thème

La plupart du temps, la création d'un jeu arrive comme une étincelle. Nous avons une idée sur la façon dont le jeu devrait fonctionner et nous commençons à créer. La création concrète de graphismes dans les premières étapes du développement peut souvent mener à des problèmes, car le jeu évolue durant le développement. Habituellement, cela aide de créer un moteur de jeu et le cœur fonctionnel avant de commencer le travail sur les graphismes. Une fois que vous savez comment le jeu se joue, il est beaucoup plus facile de trouver un thème visuel qui corresponde à son ensemble.

1.1.c Consistance

La création d'un aspect consistant est un élément clé pour acquérir une bonne expérience de jeu. Cela commence avec l'icône, l'écran de démarrage et

continue jusqu'à l'écran de gameover. Les erreurs les plus courantes sont :

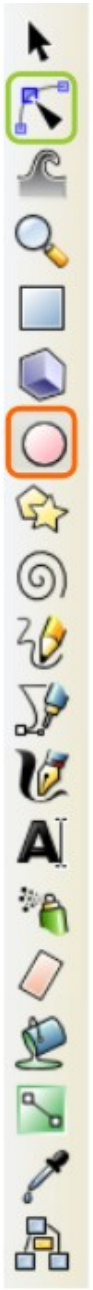
- la sur-utilisation de polices : gardez-en juste deux ou trois pour l'ensemble de l'interface utilisateur (à moins que la police ne soit utilisée dans les images pour les boutiques/présentations/etc.) ;
- Les changements radicaux de lumière et de contraste : gardez les écrans sur un même niveau – vous pouvez progresser à travers le royaume des couleurs – par exemple en commençant avec un jeu fade qui devient de plus en plus coloré pour les boss ou scènes épiques ;
- Les effets Photoshop : ils sont amusants mais la plupart des « artistes » pensent que plus vous en utilisez, mieux l'image sera... Ma suggestion est de limiter votre utilisation à quelques effets que vous pouvez réutiliser et faire varier ;
- l'éclairage : regardez vos écrans et imaginez les sources d'éclairage nécessaires pour créer les effets de lumières, ombres des éléments du jeu et de l'interface utilisateur... C'est effrayant le nombre de fois où vous trouvez des lumières utilisées aléatoirement pour les objets qui apparaissent sur un même écran.

1.1.d Perte de focus

Il est très facile de se perdre avec les graphismes aussi bien qu'avec le code. Nous avons tous tendance à nous concentrer sur les éléments que nous aimons, tout en négligeant ceux que nous n'aimons pas. Généralement, un bon exemple est le système de menus/interfaces utilisateur. Ceux-ci sont habituellement implémentés vers la fin du développement tout en ayant une motivation très basse. Mais au final, c'est l'une des premières choses que le joueur verra et ces éléments peuvent grandement définir l'aspect du jeu.

2 Commençons

Le premier tutoriel repose sur Inkscape et utilise principalement l'outil de cercles (marqué en orange dans la boîte à outils sur la gauche) et l'outil de nœuds (marqué en gris).



Commencez avec un cercle.

Ajoutez un autre cercle plus petit de la même couleur et encore un, blanc, plus petit.

Passez au noir et ajoutez un autre cercle avec un cinquième cercle blanc pour le reflet de lumière.

Dupliquez les cercles de l'œil et déplacez-les de l'autre côté.

Ajoutez un petit cercle d'une couleur plus sombre au-dessus de l'œil.

Convertissez les cercles en chemins (Chemin / Objet vers chemin) et utilisez l'outil d'édition de chemin pour le déformer.

Ajoutez un autre cercle et déformez ses nœuds supérieurs et inférieurs pour dessiner une bouche.

Dupliquez la bouche et changez la couleur de remplissage à blanc.

Créez un autre petit cercle noir pour ajouter quelques détails au visage.

... et notre personnage de jeu basé sur des cercles est prêt.

L'immense avantage des graphismes vectoriels est la facilité de modification.

Après avoir changé les couleurs des cercles, redimensionnez les cercles noirs et blancs.

. . . et ajustez la position pour avoir une autre expression faciale.

Dupliquez et redimensionnez les différents éléments permet des possibilités infinies.

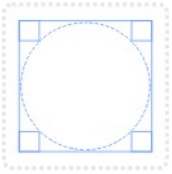
Déformez les cercles pour avoir une bouche et des antennes.

. . . ou simplement en modifiant la couleur pour changer la personnalité du personnage.

Des images plus avancées peuvent êtres créées en utilisant des dégradés et en ajoutant des versions plus claires et plus sombres de la forme pour avoir un effet cartoon basique (plus d'information sur cela plus tard).

Un résultat très similaire peut être obtenu avec un outil de dessin (par exemple avec l'outil cercle de Gimp). Le processus de travail est quelque peu différent mais tant que vous gardez les éléments sur différents calques, c'est facile de les déplacer, altérer ou modifier.

Commencez avec un canevas vide (de préférence deux fois la taille de l'image du jeu) et créez un nouveau calque.



Au lieu de dessiner un cercle, nous créons simplement un masque circulaire.

... et utilisons l'outil de remplissage pour colorier le cercle.

Dupliquez le calque et redimensionnez-le de la moitié de la taille. Placez le nouveau calque à sa position.

Créez un autre calque et déplacez-le de l'autre côté.

Dupliquez et redimensionnez les calques une nouvelle fois. Utilisez les options couleur/colorier pour changer la couleur des duplicatas à blanc.

Répétez l'opération pour créer deux autres cercles noirs plus petits et deux blancs pour les reflets des yeux.

Dupliquez les cercles noirs et coloriez-les avec une teinte marron.

Redimensionnez la bouche et ajoutez un duplicata de calque blanc. Redimensionnez et tournez les sourcils.

Voici notre personnage de jeu créé entièrement en redimensionnant et manipulant une forme simple, sans même utiliser de compétence de dessin.

Cela conclut mon premier article. J'espère que vous vous amusez à refaire quelques-uns de ces tutoriels par vous-même et jouer avec Inkscape, Gimp et autres.

*Retrouvez l'article d'**Alexandre Laurent** en ligne : lien 79*

Comment la musique a transformé le design de Rayman Legends

Emile Morel, lead game designer chez Ubisoft, a tenuune présentation sur la façon dont la musique a transformé Rayman Legends. Dans cette conférence, il nous explique comment les musiques du jeu ont influé sur le gameplay.

1 Introduction

Emile Morel, lead game designer chez Ubisoft, a tenu une présentation sur la façon dont la musique a transformé Rayman Legends. Dans cette conférence, il nous explique comment les musiques du jeu ont influé sur le gameplay.

1.1 Gamelier

Cette conférence a été organisée par **Gamelier** (lien 80), un groupe proposant quatre types d'activités :

- les présentations ;
- les workshops ;
- les playtests ;
- les microjams.

Ainsi, régulièrement, il invite des personnalités du jeu vidéo tels qu'Emile Morel pour présenter quelques aspects précis de leurs travaux.

Ce club est situé dans les locaux du Centre de Recherche Interdisciplinaire à Paris : lien 81.

1.2 Emile Morel

Emile Morel est lead game designer chez Ubisoft et a notamment travaillé sur Rayman Legends, mais encore sur Tintin et le secret de la licorne (Ubisoft), Test Drive Unlimited 2 (Eden Games) et Alone In the Dark : Inferno (Eden Games). Un de ses rôles dans le projet, a été de créer la documentation des éléments de gameplay, mais aussi de valider les tests et de faire en sorte que le jeu soit le plus parfait et amusant possible.

1.3 Rayman Legends

Rayman Legends est un jeu de plateforme 2D, reprenant les principes du premier Rayman. Toutefois, le jeu est mis au goût du jour et utilise un moteur de jeux complètement en 3D. Le jeu est sorti en août 2013 sur sept supports différents et a reçu de très bonnes critiques, notamment un score de 92 sur Metacritic (lien 82) pour la version Wii U.

Pour sa version japonaise, l'équipe d'Ubisoft a fortement travaillé en coopération avec Nintendo afin de l'adapter pour le public nippon.

2 La genèse des niveaux musicaux

2.1 Ce qu'il y avait dans Rayman Origins

2.1.a Les coffres à pattes

Coffre à pattes

L'équipe de Rayman Legends souhaitait intégrer le concept des niveaux de course contre le coffre à pattes présent dans Rayman Origins. Ces niveaux étaient courts, très techniques et il fallait les faire le plus rapidement possible. Comme la mort était courante, le redémarrage de la carte était rapide.

Les joueurs aimaient beaucoup ces niveaux, malgré la difficulté et n'hésitaient pas à essayer encore et encore pour réussir.

2.1.b Monde musical

Le second monde présent dans Rayman Origins est un monde musical où tous les éléments de gameplay produisent des sons et s'intègrent avec la musique du fond. Pour que cela ne soit pas une cacophonie, les sons ne sont pas toujours joués au moment de l'action, mais peuvent avoir un imperceptible décalage pour mieux s'intégrer à la musique.

2.1.c Conception des niveaux

Dans Rayman Origins et Rayman Legends, les niveaux peuvent être parcourus soit de façon normale lorsque le joueur découvre le niveau, soit de manière rapide en ne faisant que courir et aller le plus vite possible. Dans ce cas-là, le niveau est rythmé et les enchaînements d'appui sur les boutons d'actions sont courts.

Pour cela, le niveau doit être pensé pour qu'il soit fluide (notion de flow).

2.2 Origine de l'idée

Le studio d'Ubisoft Montpellier devait présenter les challenges sans fin de Rayman Legends où, plus le joueur était bon, plus il pouvait aller loin dans le niveau et donc, découvrir les briques de game design avancées. Mais seul le game designer travaillant sur cet aspect était capable d'atteindre ces briques et donc de montrer l'intégralité du concept. Ne pouvant pas venir jouer durant la présentation, le studio a décidé de faire une vidéo de lui en train de jouer. Pas simplement une vidéo capturant sa partie, mais tout un montage réalisé par Michel Ancel et qui avait la particularité d'être synchronisé avec la musique.

La vidéo a eu un énorme succès et immédiatement après la fin de la présentation, Michel Ancel demanda à toute l'équipe de bosser sur cette idée de niveaux synchronisés avec la musique.

2.3 Implémentation

Le principe de base revient à être un mélange entre un jeu de plateforme et un jeu de rythme. Toutefois, la principale difficulté est de faire en sorte que les actions soient toujours synchronisées avec la musique. Pour cela, les développeurs ont ajouté le principe d'une tête de lecture. Si le joueur a du retard ou autre, alors il sera recalé automatique au bon endroit correspondant au bon rythme pour la musique et les événements sur le niveau. Ce recalage est imperceptible, car il est mêlé au reste de l'action et est généralement très léger : le joueur étant obligé de courir dans ces niveaux.

2.4 Première présentation des niveaux musicaux

Le premier concept de niveau musical a dû être préparé en quelques semaines pour l'E3 2012. Pour cela, deux morceaux ont été composés une fois que le niveau a été conçu. Le premier morceau a été réalisé par Christophe Heral, compositeur de l'équipe et le second par un studio externe. C'est le morceau de Christophe qui a été retenu :

Cliquez pour visualiser la vidéo.

Ici, c'est Émile Morel qui joue sur le gamepad de la Wii U.

En plus de cette présentation durant la conférence de Nintendo, le niveau est jouable pendant les trois jours de l'événement. Le public fut conquis et adora le concept.

2.5 Création des autres niveaux musicaux

Les musiques utilisées pour les autres niveaux sont un mélange de compositions originales et de reprises de morceaux. Pour choisir les morceaux, toute l'équipe de développement a eu la possibilité d'en proposer. Toutefois, il fallait au moins que le morceau soit en rythme avec l'animation de Rayman, qu'il possède quelques variations et que le tempo soit assez rapide.

Le processus de création est plus classique : la musique est composée, puis le niveau est calé sur la musique.

Cliquez pour visualiser la vidéo.

Cette musique est une reprise de la bande originale de Rocky 3 : Eyes of the tiger. La voix que l'on entend dans le jeu est celle de Christophe Heral.

2.6 Les versions 8 bits

Les niveaux musicaux 8 bits arrivent en bonus lorsque le joueur a fini le jeu. La volonté était de refaire jouer ces niveaux très amusants, tout en proposant un challenge encore plus élevé que pour les autres cartes. Les versions 8 bits mettent l'accent sur l'aspect jeu de rythme. En fait, les effets graphiques sont tels qu'ils ne permettent plus de distinguer grand-chose du jeu.

L'idée des versions 8 bits vient de Christophe Heral ayant réalisé une version 8 bits du thème de Rayman Origins.

Cliquez pour visualiser la vidéo.

Sur ces niveaux, le public est mitigé. Certains adorent, d'autres se demandent pourquoi ils ont été mis dans le jeu.

2.7 Bonus

Les niveaux musicaux avaient été joués en live durant un press tour à San Franscisco et à Montpellier. Pendant que les journalistes jouaient au jeu, les musiciens (dont Michel Ancel et Christophe Heral) jouaient la musique et devaient aussi reproduire les sons lorsque le joueur perdait.

La musique d'introduction change à chaque mort du joueur.

3 Questions

A-t-il fallu qu'Ubisoft demande des droits pour reprendre les musiques ?

Oui, cela est obligatoire chaque fois qu'un studio de développement utilise une musique sous licence, même si cela est dans le cadre d'une reprise. Toutefois, Ubisoft est habitué à ce genre de négociation, notamment pour les musiques de Just Dance.

Est-ce que l'équipe a eu une énorme pression pouvant altérer le processus créatif dû à la popularité et l'attente des niveaux musicaux ?

Non, car ces niveaux reposent sur une base très solide et l'engouement global était plutôt une source de motivation pour l'équipe.

Quelle est la taille de l'équipe de développement de Rayman Legends ?

Au plus fort de la production, l'équipe était constituée de 80 personnes.

Pourquoi ne pas faire que des niveaux musicaux ?

Avoir une différence permet de mettre en valeur les niveaux musicaux. De plus, ils sont utilisés comme récompense pour les joueurs finissant un monde.

Est-ce que ce sont les mêmes personnes qui ont fait les niveaux musicaux ?

Oui, tous les niveaux ont été faits par un unique level designer. Celui-ci se devait d'avoir une bonne oreille musicale.

Comment sont gérés les playtests à Ubisoft Montpellier ?

Les playtests démarrent très tôt dans la production du jeu. Le recrutement est réalisé directement sur place. Rayman Legends étant aussi un jeu très amusant à travers son mode coopératif. Il pouvait y avoir un père et son fils venant jouer durant les playtests.

Quels sont vos conseils pour garder l'équipe motivée ?

Il faut que les membres de l'équipe puissent s'accaparer le chantier. De plus, il est nécessaire de toujours avoir un membre moteur, toujours très motivé (Michel Ancel pour le cas de Rayman Legends), qui puisse remotiver l'équipe lorsque nécessaire.

Est-ce qu'UbiArt sera disponible au grand public ?

C'est le souhait de Michel Ancel. Toutefois, UbiArt est actuellement trop compliqué, car il a été développé en parallèle de son utilisation dans les jeux et manque donc de quelques raffinements.

Que faut-il connaître pour être game designer ?

Il faut maîtriser PowerPoint, Excel. Il est important d'avoir un bon niveau en anglais, de savoir se servir d'un moteur et de comprendre les contraintes des autres membres de l'équipe. Savoir motiver et avoir une bonne culture en jeu vidéo est aussi un plus.

*Retrouvez l'article d'**Alexandre Laurent** en ligne : lien 83*

Reseau

Les derniers tutoriels et articles

La technologie RNIS

Cet article est une introduction à la technologie des Réseaux Numériques à Intégration de Services (RNIS). Il présente uniquement les caractéristiques, les modes de fonctionnement et les protocoles utilisés dans les échanges de données. Si cette technologie n'est plus du tout en vogue, il est toujours utile d'en connaître les propriétés, notamment pour interfacer les infrastructures de téléphonie historique avec les réseaux IP.

1 Copyright et Licence

1.1 Méta-information

Cet article est écrit avec DocBook XML (lien 84) sur un système Debian GNU/Linux (lien 85). Il est disponible en version imprimable au format PDF : lien 86.

2 Qu'est-ce qu'un réseau RNIS ?

L'architecture des Réseaux Numériques à Intégration de Services (RNIS) a été conçue pour associer la voix, les données, la vidéo et tout autre application ou service. Cette architecture peut être vue comme une évolution des réseaux téléphoniques analogiques historiques ou Plain Old Telephone System (POTS). Les réseaux RNIS bande de base fournissent des services à faible débit : de 64 kbps à 2 Mbps. La technologie ATM (Asynchronous Transfer Mode) dédiée aux réseaux grandes distances (WAN) faisait à l'origine partie des définitions RNIS sous la dénomination RNIS large bande pour les services à haut débit : de 10 Mbps à 622 Mbps.

Avec les réseaux RNIS, les sites régionaux et internationaux de petite taille peuvent se connecter aux réseaux d'entreprises à un coût mieux adapté à la consommation réelle qu'avec des lignes spécialisées. Les liaisons à la demande RNIS peuvent être utilisées soit pour remplacer les lignes spécialisées, soit en complément, pour augmenter la bande passante ou assurer une redondance. Avec ces mêmes liaisons, les sites ou les utilisateurs distants, peuvent accéder efficacement aux ressources critiques à travers l'Internet en toute sécurité.

De nos jours, les mêmes types d'accès réseau sont assurés avec d'autres technologies offrant davantage de performances. Dans le monde des réseaux filaires, les technologies xDSL dominent largement et dans le monde des radio-communications les capacités de transfert de données offertes par les réseaux 3G et 4G dépassent facilement les débits des réseaux RNIS.

3 Le développement des réseaux RNIS

L'Union Internationale des Télécommunication (ITU : lien 87) a défini la technologie RNIS comme un réseau fournissant une connectivité numérique de bout en bout avec une grande variété de services.

Deux caractéristiques importantes des réseaux RNIS les distinguent des réseaux téléphoniques traditionnels :

— les connexions sont numériques d'une extrémité à l'autre ;
— RNIS définit un jeu de protocoles d'interface utilisateur/réseau standard. De cette façon, tous les équipements RNIS utilisent les mêmes connexions physiques et les mêmes protocoles de signalisation pour accéder aux services.

RNIS combine la large couverture géographique d'un réseau téléphonique avec la capacité de transport d'un réseau de données supportant simultanément la voix, les données et la vidéo.

En France, les connexions RNIS sont disponibles sous la dénomination commerciale Numéris®. Le réseau national de télécommunication a été entièrement numérisé et les protocoles d'accès implantés par France Télécom™ sont conformes au standard Euro-ISDN publié par l'ETSI (lien 88) et l'ITU.

4 Comment fonctionne un réseau RNIS ?

Dans un réseau téléphonique analogique, une boucle sur une paire torsadée de fils de cuivre entre le commutateur central de la compagnie de télécommunication et l'abonné supporte un canal de transmission unique. Ce canal ne traite qu'un seul service simultanément : la voix ou les données. Avec un Réseau Numérique à Intégration de Services, la même paire torsadée est divisée en plusieurs canaux logiques.

4.1 Les canaux logiques RNIS

RNIS définit deux types de canaux logiques que l'on distingue par leurs fonctions et leurs débits.

— Les canaux B transmettent à un débit de 64 kbps, en commutation de circuits ou de paquets, les informations utilisateur : voix, données, fax. Tous les services réseau sont accessibles à partir des canaux B.
— Les canaux D transmettent à un débit de 16 kbps en accès de base (lien 89) et 64 kbps en accès primaire (lien 90). Ils supportent les informations de signalisation : appels, établissement des connexions, demandes de services, routage des données sur les canaux B et enfin libération des connexions. Ces informations de signalisation ont été conçues pour cheminer sur un réseau totalement distinct des canaux B. C'est cette signalisation hors bande qui donne aux réseaux RNIS des temps d'établissement de connexion rapides (environ quatre secondes) relativement aux réseaux analogiques (environ 40 secondes). Il est aussi possible de transmettre des données utilisateur à travers les canaux D (protocole X.31b), mais comme le débit de ces canaux est limité, ce type d'utilisation est rare.

4.2 Les interfaces standards RNIS

Une interface d'accès à un réseau RNIS est une association de canaux B et D. Il existe deux interfaces standards. Elles correspondent à deux catégories d'utilisation distinctes :

— résidentielle : utilisation simultanée des services téléphoniques et d'une connexion Internet ;
— professionnelle : utilisation d'un commutateur téléphonique (PABX) et/ou d'un routeur d'agence.

Dans les deux cas, le nombre de canaux utilisés peut varier suivant les besoins. Le débit maximum étant fixé par le type d'interface :

— accès de base : l'accès de base ou Basic Rate Interface (BRI) comprend deux canaux B et un canal D pour la signalisation : 2B+D.
— accès primaire : l'accès primaire ou Primary Rate Interface (PRI) comprend 30 canaux B et un canal D à 64 kbps en Europe : 30B+D. Aux États-Unis et au Japon la définition est différente : 23B+D. Seule la protection des marchés explique les différences de définition entre l'Europe, les États-Unis et le Japon. Cet accès est l'équivalent RNIS des liaisons T1/E1 à 2,048 Mbps et 1,544 Mbps.

4.3 L'adaptation des débits

Les équipements non RNIS n'ont pas nécessairement des débits compatibles avec la définition du canal B : 64 kbps. Dans ce cas, les adaptateurs de terminal (TA) réalisent une adaptation en réduisant le débit effectif du canal B jusqu'à une valeur compatible avec le dispositif non RNIS.

Il existe deux protocoles de gestion d'adaptation : V.110 très utilisé en Europe et V.120 aux États-Unis. Ces deux protocoles gèrent les transmissions synchrones et asynchrones. Le protocole V.110 peut fonctionner avec un téléphone cellulaire GSM, par exemple. C'est au prestataire de téléphonie cellulaire de fournir la passerelle RNIS/V.110.

4.4 L'allocation dynamique de bande passante

La bande passante dynamique ou l'allocation de canaux est obtenue par l'agrégation des canaux B.

On obtient ainsi une bande passante maximale de 128 kbps pour l'accès de base (BRI) (lien 91) et de 1,920 Mbps (30 canaux à 64 kbps) pour l'accès primaire en Europe (lien 92).

Cette fonctionnalité permet d'adapter le débit et donc le coût de communication aux besoins effectifs pour les flux entrants et sortants. Suivant les heures de la journée ou les jours de la semaine, les besoins de connectivité varient fortement. Il est possible que le coût forfaitaire d'utilisation d'une ligne spécialisée soit supérieur au coût en temps de communication d'une liaison RNIS, lorsque cette dernière utilise correctement la bande passante à la demande en ouvrant/fermant les connexions aux heures choisies.

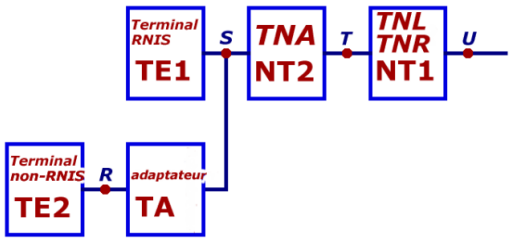
Il existe deux techniques pour agréger les canaux B appelées bonding et bundling.

— Le bonding travaille au niveau 1 (couche physique) du modèle OSI. Il assure une synchronisation au niveau bit. Cette technique nécessite donc un matériel spécifique. Elle est surtout utilisée dans les équipements dédiés de visioconférence et très peu dans les équipements de réseaux de données.

— Le bundling est une technique générique qui travaille au niveau 2 (couche liaison) du modèle OSI. Dans le cas d'une connexion RNIS, elle permet d'ouvrir simultanément plusieurs canaux B entre deux systèmes. Le standard ML-PPP décrit comment séparer, recombiner et séquencer des datagrammes sur plusieurs canaux B, pour créer une connexion logique unique. Ce standard est dédié au protocole Linux PPP HOWTO (lien 93), le protocole de niveau liaison le plus utilisé avec le modèle TCP/IP pour les accès téléphoniques. Les documents RFC1717 (lien 94) The PPP Multilink Protocol (MP), puis RFC1990 (lien 95) The PPP Multilink Protocol (MP) décrivent le logiciel de niveau liaison associé à PPP.

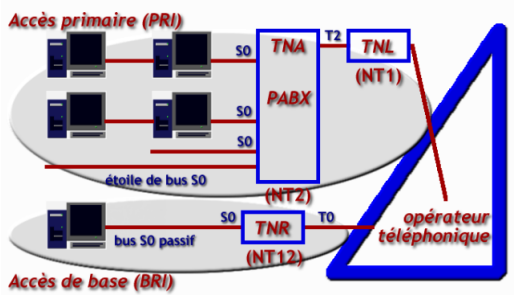
5 Dispositif de connexion RNIS

La configuration physique vue du côté de l'utilisateur RNIS est divisée en groupes fonctionnels séparés par des points de référence. Un groupe fonctionnel est une association particulière d'équipements qui assurent un ensemble de fonctions RNIS. Les points de référence sont les limites qui séparent les différents groupes fonctionnels. À chacun de ces points de référence correspond une interface standard à laquelle les fournisseurs d'équipements doivent se conformer. Ces interfaces standards ont aussi pour but de permettre à l'utilisateur de choisir son équipement librement.



— R, S, T, U : points de références.
— TNL-TNR/NT1 : Terminal Numérique de Ligne-Terminal Numérique de Réseau/Network Termination 1.
— TNA/NT2 : Terminal Numérique d'abonné/-Network Termination 2.
— Terminal RNIS/TE1 : Terminal Equipment 1.
— Adaptateur/TA : Terminal Adapter.
— Terminal non RNIS/TE2 : Terminal Equipment 2.

Le schéma ci-dessus fait apparaître les dénominations anglo-saxonnes et françaises (en italique). Suivant la répartition entre opérateurs téléphoniques de la prise en charge des liaisons, il peut y avoir des regroupements entre groupes fonctionnels. En France le « dégroupage » est devenu possible avec l'ouverture du marché à de nouveaux opérateurs téléphoniques.



Topologies des installations RNIS :

— pour un accès de base (BRI), l'appellation est Terminal Numérique de Réseau. Le TNR comprend les deux groupes NT1 et NT2. Il n'existe pas de TNA. En France, les offres Numéris DUO® et Numéris Itoo® de France Télécom™ utilisent des TNR-G qui répondent à ce critère ;
— pour un accès primaire (PRI), l'appellation est Terminal Numérique de Ligne.

5.1 Terminal Numérique de Réseau ou de Ligne

Selon la définition, le groupe fonctionnel NT1 est la liaison physique et électrique entre le commutateur central de l'opérateur téléphonique et le réseau

de l'abonné. Ce groupe supporte les interfaces usager/réseau avec de multiples canaux à multiplexage temporel (Time-Division Mutiplexing - TDM). La connexion n'autorise que des équipements RNIS.

5.2 Terminal Numérique d'Abonné

Le groupe fonctionnel NT2 n'est utilisé que pour les accès primaires. Ce groupe possède de nombreuses fonctions de commutation de circuits ou de paquets avec plusieurs connexions de bus S0. En règle générale, ce groupe correspond à un commutateur local (PABX) opérant au niveau réseau.

5.3 Terminal RNIS

Un terminal RNIS (TE1) possède une interface S0 sans adaptation. Typiquement, les ordinateurs avec des modems internes RNIS sont des terminaux RNIS.

5.4 Adaptateur

Le rôle de l'adaptateur est de rendre compatible le débit du terminal non RNIS avec celui du canal B du bus S0 : 64 kbps. Typiquement, les modems externes sont appelés Terminal Adapters.

5.5 Terminal non RNIS

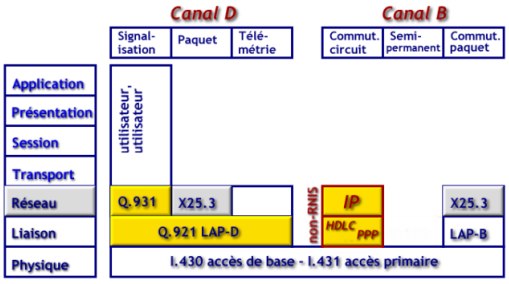
Un terminal non RNIS (TE2) ne possède pas d'interface S0 directe tout comme les dispositifs utilisant des ports série, des bus USB, etc.

5.6 Points de référence

- Le point U est placé entre le groupe NT1 et la boucle de transmission de l'opérateur téléphonique qui fournit une liaison bidirectionnelle (full-duplex) entre l'abonné et le commutateur central sur deux fils. L'interface U n'est utilisée qu'en Amérique du nord.
- Le point T est placé entre le groupe NT2 qui possède des fonctions de niveaux 1 à 3 et le groupe NT1 qui ne possède que des fonctions de niveau 1. C'est le point de connexion minimal entre l'abonné et l'opérateur. Il existe plusieurs appellations suivant les types d'accès :
  - T0 : accès de base (BRI) 2B+D ;
  - T2 : accès primaire (PRI) 30B+D. En France, les accès T2 sont déclinables en 15, 20, 25 et 30 canaux B.
- L'interface S peut être assimilée à un bus passif pouvant supporter huit terminaux (TE) en série sur le même câble. Dans ce cas, chaque canal B est affecté à un terminal particulier pour la durée d'un appel.
- Le point R est la limite conceptuelle entre le terminal non RNIS et l'adaptateur.

6 Les protocoles RNIS

Organisation des protocoles RNIS dans la modélisation OSI.



Protocoles RNIS - vue complète : lien 96

6.1 Couche Physique (1)

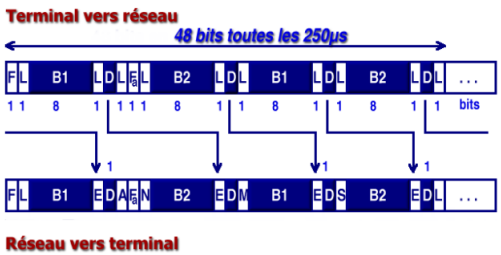
La couche physique (ou niveau 1) est identique pour les canaux B et D qui sont multiplexés pour composer un accès de base ou un accès primaire. Pour la suite de la présentation, nous prenons l'exemple de l'accès de base 2B+D.

La structure de la trame est composée de 48 bits répétés toutes les 250 µs, soit un débit total de 192 kbps. La distribution des débits entre les canaux B et le canal D est réalisée par mutiplexage. Chaque trame contient :

- deux octets pour le premier canal B (B1) ;
- deux octets pour le second canal B (B2) ;
- quatre bits pour le canal D répartis sur la trame.

6.1.a Formats de trames

Les formats de trames dépendent du sens de transmission entre le terminal RNIS (TE) et le Terminal Numérique de Réseau (TNR).



Formats des trames RNIS - vue complète : lien 97

- F : Framing bit, synchronisation de trame.

— L : DC-balancing bit, équilibrage de la composante continue.

— E : D-channel Echo bit, bit d'écho du canal D.

— A : Activation bit, bit d'activation du terminal.

— Fa : auxiliary Framing bit, synchronisation auxiliaire.

— N : opposé du bit de synchronisation auxiliaire.

— M : Multiframing bit.

— B1, B2 : bits des canaux B ; 16 bits par trame.

— D : bits du canal D ; 4 bits par trame.

— S : Spare bits, bits disponibles.

6.1.b Méthode d'accès au bus S0

Un canal B est toujours dédié à un terminal alors que le canal D est partagé entre tous les terminaux connectés sur le bus S0. La méthode d'accès au canal D employée par RNIS est voisine de celle d'Ethernet. Son appellation est : CSMA/CR Carrier Sense Multiple Access with Contention Resolution.

Une station (ou un terminal) qui n'a rien à transmettre émet continuellement des niveaux '1' logiques (no signal). Le nombre de niveaux '1' logiques (de 8 à 11) correspond à une priorité prédéfinie.

— Les services téléphoniques sont prioritaires sur les autres types de services.

— Les informations de signalisation sont prioritaires sur les autres types d'informations.

Une station prête à émettre scrute le bit E des trames provenant du réseau. Le bit E, émis par le réseau (NT), est l'écho du bit D précédemment transmis par le terminal (TE). Si une station (ou un terminal) détecte un bit E différent du bit D émis, il y a collision. Cette station stoppe immédiatement son émission. Cette technique simple de résolution des collisions garantit qu'une seule station émet sur le canal D simultanément. Après une transmission complète sur le canal D, la priorité du terminal concerné est réduite. Il doit détecter davantage de niveaux '1' logiques pour pouvoir émettre à nouveau. La priorité d'un terminal ne peut être augmentée avant que tous les autres terminaux du bus S0 aient cessé d'émettre.

6.2 Couche Liaison (2)

Comme indiqué ci-avant, les modes de fonctionnement des canaux B et D sont très différents.

6.2.a Canal B

Il existe trois modes de connexion : commutation de circuits, mode semi-permanent et commutation de paquets.

— Commutation de circuits : le circuit est établi, maintenu et libéré en utilisant la signalisation du canal D. Les données utilisateur sont échangées sur les canaux B avec les protocoles utilisateur.

— Mode semi-permanent : le circuit est établi entre les utilisateurs et le réseau pour une durée délimitée ou non. Une fois le circuit établi, le canal D n'est plus nécessaire pour la signalisation.

— Commutation de paquets : dans ce cas, une connexion en mode commutation de circuits doit être établie entre l'abonné RNIS et un noeud du réseau à commutation de paquets sur le canal B. Cette connexion en mode commutation de circuits implique l'utilisation de la signalisation du canal D. Le réseau à commutation de paquets peut être partiellement RNIS. RNIS peut donc fournir un service de commutation de paquets sur les canaux B (protocole X.31a).

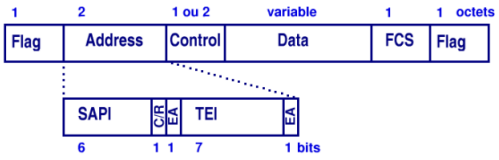
Pour la transmission sur les réseaux de données, notamment Internet, la commutation de circuits est le mode de connexion le plus largement adopté. Ce guide s'appuie sur ce mode de transmission.

6.2.b Canal D

Il existe trois types de services sur le canal D : signalisation, commutation de paquets et télémétrie. Ces services sont tous intégrés dans le même protocole de niveau 2 appelé LAP-D. Ce protocole est voisin de la normalisation X25.2 : trames au format HDLC (High-Level Data Link Control) et protocole LAP-B (Link Access Protocol - Balanced Mode).

Le rôle des trames HDLC est de contrôler la liaison de données entre le Terminal Numérique de Réseau (TNR) et le Terminal RNIS (TE1).

Le protocole LAP-D est normalisé par l'ITU : spécifications Q.920 et Q.921. La principale différence entre les protocoles LAP-B et LAP-D réside dans l'adressage (champ Address). À partir des champs TEI/SAPI, l'adressage LAP-D permet de gérer les liaisons multipoints : plusieurs services pour une même interface ou diffusion d'un message vers toutes les interfaces du bus S0. Les champs Flag et Control sont identiques au format HDLC. La taille maximale de trame est limitée à 260 octets.



Trame HDLC LAP-D - vue complète : lien 98

Champs de la trame HDLC

— *Flag* : délimiteur de trame = 7Eh ou 01111110 en binaire.

— *Address* : adressage RNIS : services & terminaux. Voir Adressage LAP-D ci-après.

— *Control* : contrôle des appels. Voir Contrôle de connexion ci-après.

— *Data* : données de la trame.

— *Frame Check Sequence, FCS* : somme de contrôle. Vérification de la cohérence de la trame.

— *Flag* : délimiteur de trame identique au premier champ.

De même que pour la méthode d'accès au média, les fonctionnalités d'adressage sont analogues entre Ethernet et LAP-D. Le champ TEI correspond au champ MAC de la trame Ethernet IEEE 802.3. La valeur TEI n'occupe que sept bits au lieu de six octets. Contrairement à Ethernet, cette valeur est attribuée lors de la connexion et elle n'est pas censée être unique pour la totalité des réseaux téléphoniques. Les champs SAPI et C/R correspondent aux champs SSAP et DSAP de la norme IEEE 802.2.

Adressage LAP-D

— Service Access Point Identifier, SAPI : identification des services fournis à la couche réseau (niveau 3).

— Command/Response, C/R : indique si la trame est une commande ou une réponse.

— Terminal End-point Identifier, TEI : identification unique du terminal (de l'interface) ou diffusion à tous les terminaux (valeur 127 - tous les bits à 1).

— End Address, EA : extension d'adresse : valeur 0 au premier octet et 1 au second.

La gestion des appels est assurée par le champ Control qui occupe un ou deux octets suivant le type de contrôle.

Contrôle de connexion

— Information, (I) : les trames I sont utilisées pour le transfert d'informations sur les services de niveau 2 utilisés par le niveau 3. Elles contiennent en plus les numéros de séquences. Elles occupent donc deux octets. On trouve un exemple de ce type de trames après le choix du canal B au début d'une séquence de connexion.

— Supervision, (S) : les trames S sont un ensemble de commandes de supervision de liaison. Elles contiennent les numéros d'acquittement en plus des commandes. Elles occupent donc deux octets.Non numéroté, (U) : les trames U ne sont pas numérotées. Elles occupent un seul octet. On ne peut donc pas contrôler leur séquencement. Elles utilisent un jeu de commandes (ou questions/réponses) pour l'établissement et la libération des liaisons de données.

— La commande Set Asynchronous Balanced Mode Extended (SABME) est une demande d'initialisation de liaison de données avec remise à zéro des numéros de séquence.

— La commande Unnumbered Acknowledgement (UA) est un acquittement qui indique que le terminal est disponible pour l'établissement d'une liaison de données.

— Les commandes Unnumbered Information (UI) jouent un rôle très important. Elles assurent l'échange d'informations sans connexion : messages d'établissement et gestion des TEI. Leur fonctionnement est analogue à celui du protocole PPP au niveau 3 pour l'attribution des adresses IP lors d'une connexion téléphonique.

— La commande Disconnected Mode (DM) indique que le terminal est déconnecté.

— La commande Disconnect (DISC) indique la libération de la liaison de données et la remise à zéro des numéros de séquence.

— La commande Frame Reject (FRMR) est un rejet de trame dû à une erreur sur la validité d'un ou plusieurs champs : information non valide, numéro de séquence erroné ou longueur de trame incorrecte. On peut comparer cette commande à l'émission du JAM sur les réseaux Ethernet.

6.3 Couche Réseau (3)

Comme pour le niveau précédent, les modes de fonctionnement des canaux B et D sont très différents.

6.3.a Canal B

Il n'existe pas de protocole RNIS spécifique au niveau 3 pour les canaux B. Suivant le mode de commutation choisi au niveau 2, on peut utiliser différents protocoles.

— La commutation de circuits étant le mode d'accès privilégié pour les connexions à Internet, on retrouve donc les protocoles du modèle TCP/IP au niveau réseau.

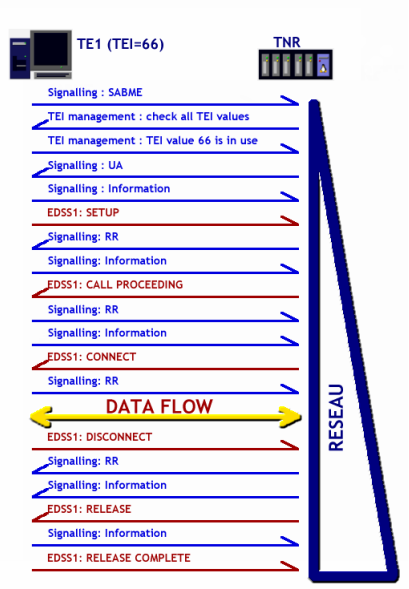
— Les protocoles X.25 et X.75 sont utilisables pour accéder aux réseaux à commutation de paquets. Le protocole X.75 est voisin d'X.25. Il est dédié aux services internationaux de commutations de paquets : mode STE-STE (Signalling Terminal Equipement). Il existe deux situations types d'accès aux réseaux à commutation de paquets :

— accès à un réseau public X.25 à partir d'un raccordement physique sur un canal B RNIS. Ce cas de figure est défini par le protocole X.31a,
— utilisation des téléservices au-dessus du protocole réseau ISO-8208. ISO-8208 en mode DTE-DTE est très similaire à X.25. Le service EUROfile transfer est un bon exemple de ces téléservices.

6.3.b Canal D

Le protocole de niveau 3 ou protocole D gère principalement l'établissement, le maintien et la libération des connexions. Il peut aussi assurer le transfert d'informations (protocole X.31b) et des compléments de services.

Les spécifications ITU I.450/Q.930 et I.451/Q.931 définissent les messages de gestion des connexions. Voici un exemple de dialogue en commutation de circuit : protocole Q.921 en bleu et protocole Q.931 en rouge.



Exemple de connexion - vue complète : lien 99

7 Référence sur la technologie RNIS

Le document FAQ ISDN/Numéris - Version Juin 2000 de Franck Brunel est une ressource de qualité sur la technologie RNIS : lien 100.

Retrouvez l'article de ***Philippe Latu*** *en ligne : lien 101*

Modélisation en couche des protocoles réseau

La modélisation du fonctionnement des réseaux électroniques de communications a toujours fait l'objet de grandes luttes d'influence entre les organismes de normalisation, les compagnies de télécommunications et les constructeurs. Avec l'avènement de l'Internet, un modèle contemporain faisant la synthèse entre les modèles de référence historiques OSI et TCP/IP s'est imposé. L'objectif de cet article est d'introduire les concepts de modélisation, de présenter les deux modélisations dominantes et le « dénominateur commun » qui en est issu.

1 Copyright et Licence

1.1 Métainformation

Cet article est écrit avec DocBook XML (lien 102) sur un système Debian GNU/Linux (lien 103). Il est disponible en version imprimable au format PDF : lien 104.

Le ton de cet article est volontairement « polémique ». L'objectif pédagogique est de susciter la réaction et, pourquoi pas, la réflexion sur un sujet souvent jugé trop académique. Une bonne compréhension de la genèse des différents modèles d'interconnexion des réseaux de télécommunications permet de construire une analyse critique des évolutions actuelles.

2 Modélisation des réseaux de télécommunications

Si l'utilisation des connexions aux réseaux de télécommunications a explosé avec le développement de l'Internet, la conception des techniques de connexion a débuté dans les années 1960.

À cette époque, comme le nombre de fournisseurs d'équipements informatiques était réduit, chacun a développé « sa » solution de connexion. Les difficultés sont très vite apparues lorsque les utilisateurs ont eu besoin d'interconnecter des systèmes hétérogènes distants.

Aujourd'hui, la très grande majorité des interconnexions utilise des équipements et des réseaux acquis auprès de fournisseurs différents. Pour parvenir à ce résultat, il a fallu harmoniser les modes d'interconnexion. Les modélisations sont les outils essentiels de cette harmonisation.

Pendant la guerre froide, l'ARPANET, l'ancêtre militaire de l'Internet, a été développé à partir de 1969 pour maintenir les communications entre les centres névralgiques du continent nord-américain face aux menaces d'attaques nucléaires. Même si ces conditions d'utilisation sont très éloignées du contexte actuel, l'interconnexion de systèmes hétérogènes était une nécessité impérieuse pour le Département d'État des États-Unis.

Les protocoles TCP et IP qui dominent les réseaux de télécommunications contemporains ont été conçus pour répondre à ces objectifs d'harmonisation des échanges d'informations entre systèmes différents. Sans une modélisation cohérente des communications, il serait impossible de parler de quadruple-play : données, voix, vidéo et mobilité sur des réseaux aussi hétérogènes que les radiocommunications, les réseaux filaires en cuivre ou en fibre optique.

2.1 Classification

Historiquement, c'est la distance entre les équipements à connecter qui a constitué le premier critère de classement des réseaux de télécommunication.

Ce critère est fondé sur le mode de transport de l'information. Même si c'est de moins en moins vrai, on part du principe que l'on n'emploie pas les mêmes techniques pour véhiculer des données d'une pièce à l'autre ou d'un continent à l'autre.

Distance	Acronyme	Type de réseau
jusqu'à 25 mètres	PAN	Réseau local « domestique » : Personal Area Network
jusqu'à 10 km	LAN	Réseau local : Local Area Network
jusqu'à 50 km	MAN	Réseau métropolitain : Metropolitan Area Network
jusqu'à 1000 km	WAN	Réseau longue distance : Wide Area Network
jusqu'à 40 000 km	Internet	Réseau mondial

Tableau 1. classification des réseaux

2.2 Modélisations en couches

La technique usuelle en informatique pour résoudre un problème complexe consiste à le découper en problèmes simples à traiter. L'interconnexion réseau étant un problème complexe, on a donc abouti à des traitements séparés par niveaux ou couches. La fonction de chaque couche est de fournir des services à son homologue de niveau supérieur en occultant ses traitements propres.

Entre deux équipements, chaque couche dialogue à son niveau à l'aide d'un protocole. Pour l'ensemble des couches utilisées lors d'une communication réseau, on parle de pile de protocoles.

2.3 Concepts communs aux modélisations

Il existe un certain nombre de concepts communs liés aux modélisations en couches. Ces concepts sont implémentés dans les protocoles. Les protocoles peuvent être vus comme une sélection des traitements possibles au niveau de chaque couche. Voici une présentation succincte de ces concepts :

2.3.a L'adressage

Pour que chaque couche puisse reconnaître ses pairs sur les autres systèmes connectés au réseau, il est nécessaire de recourir à un adressage. Le rôle d'une adresse est d'identifier sans ambiguïté un hôte du réseau. Les mécanismes d'adressage jouent un rôle essentiel dans l'acheminement de l'information.

Il existe de très nombreux exemples d'utilisation de mécanismes d'adressage uniques ou multiples dans les réseaux :

- L'analogie la plus usuelle est fournie par le courrier papier qui est routé par le service postal en fonction de l'adresse du domicile suivant un protocole qui utilise le code postal, le type de voie, etc.
- Le courrier électronique est acheminé à partir d'une adresse composée du nom d'utilisateur (partie gauche) et d'un nom de domaine (partie droite).
- Un téléphone mobile met en œuvre plusieurs mécanismes d'adresses simultanément. Il est repéré dans une cellule par son International Mobile Equipment Identity ou code IMEI et les communications utilisent le numéro de l'abonné. C'est aussi sur le format du numéro de téléphone que des décisions d'acheminement sont prises (opérateur, zone géographique, etc.).
- Pour un hôte connecté à l'Internet, son adresse affectée par le protocole IP identifie cet hôte de façon unique. Les adresses IP ne sont généralement pas les seules utilisées dans un même système. On retrouve souvent une adresse utilisée pour repérer un hôte dans un réseau local ou dans la zone de couverture radio du réseau sans fil.

2.3.b Le routage

Les protocoles de chaque couche prennent leurs décisions d'acheminement de l'information à partir des adresses et des itinéraires disponibles. La technique d'acheminement d'une information à travers de multiples circuits de communications est appelée routage.

2.3.c Le contrôle d'erreur

Les circuits de communications n'étant pas parfaits, il est nécessaire de mettre en œuvre des mécanismes de contrôle d'erreur. Suivant le niveau de traitement de chaque couche, ces contrôles d'erreur sont pris en charge par les protocoles de chaque niveau. Au niveau le plus bas, on contrôle que le

nombre de bits reçus correspond bien au nombre de bits émis sur un média (paire cuivre, fibre optique, canal hertzien). À un niveau plus élevé, on contrôle le séquencement de l'acheminement de blocs d'informations. Si une suite de blocs est émise dans un ordre donné, ces mêmes blocs peuvent parvenir dans le désordre à l'autre extrémité d'un réseau étendu.

2.3.d Le contrôle de flux

Tous les systèmes n'ayant pas les mêmes capacités de traitement, il faut éviter que les hôtes les mieux dotés mobilisent à leur seul usage les circuits de communications. De la même façon, il faut éviter qu'un émetteur ne sature l'interface d'un récepteur plus lent. Les solutions à ces problèmes peuvent être complexes. Généralement, les protocoles implémentent des mécanismes de notification qui permettent de contrôler qu'un récepteur a bien traité l'information qui lui est destinée. On parle alors de contrôle de flux.

2.3.e Le (dé)multiplexage

Les routes empruntées par les circuits de communications dépendent de la topographie. L'interconnexion des réseaux entre les continents passe par un nombre limité de circuits appelés dorsales (backbones). La transmission de l'information sur les dorsales utilise les fonctions de multiplexage (temporel ou fréquentiel) à l'émission et de démultiplexage à la réception. Ces fonctions permettent de véhiculer plusieurs flux distincts sur un même circuit.

2.4 Commutation de circuits ou commutation de paquets

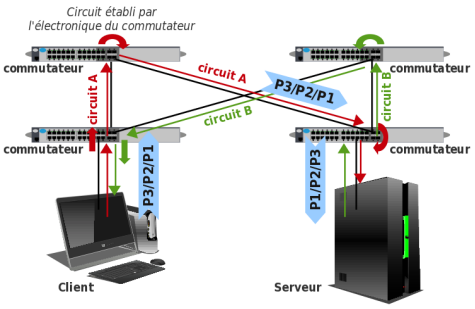
Dans les réseaux de télécommunications contemporains on retrouve deux techniques de commutation distinctes. Ces techniques peuvent se croiser dans la description des couches des modélisations et dans les technologies d'implémentation des protocoles. Ainsi, dans un réseau local, on peut très bien utiliser une commutation de circuits avec la technologie Ethernet au niveau liaison et utiliser un réseau à commutation de paquets avec le protocole IP.

2.4.a Commutation de circuits

Cette technique consiste à commuter des circuits physiques ou virtuels pour que deux hôtes du réseau puissent communiquer comme s'ils étaient connectés directement l'un à l'autre. Voici deux exemples classiques de ce type de commutation.

— Sur un réseau téléphonique filaire lors de l'émission d'un nouvel appel en composant un numéro d'abonné, les commutateurs téléphoniques établissent un circuit unique entre les deux combinés. Une fois la communication établie, les échantillons de voix transitent séquentiellement sur ce circuit.

— Sur un réseau local utilisant des commutateurs Ethernet, une fois les tables de correspondance entre les adresses physiques des hôtes constituées, les hôtes peuvent communiquer entre eux via un circuit unique établi par l'électronique des commutateurs.



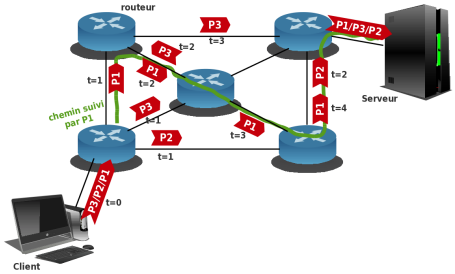
Dans la figure ci-dessus, les paquets P1, P2 et P3 sont nécessairement reçus dans l'ordre dans lequel ils ont été émis.

Si un circuit de communication est rompu, toutes les données présentes sur ce circuit sont perdues et toute communication est impossible tant qu'un nouveau circuit n'a pas été établi.

Commutation de paquets

— Avec cette technique, les informations découpées en paquets de taille limitée peuvent emprunter des itinéraires différents en fonction de l'état de l'interconnexion réseau entre deux points.

— Le protocole IP, utilisé au niveau réseau de la modélisation Internet, est l'exemple le plus connu d'exploitation de la commutation de paquets.



Les caractéristiques de ces deux types de commutation sont adaptées à différents besoins. Avec la commutation de circuits, la constitution d'un circuit unique de bout en bout permet de conserver la séquence des informations émises et la réservation de bande passante évite la congestion. Avec la commutation de paquets, la tolérance aux pannes et l'optimisation de l'utilisation des canaux de communication sont bien meilleures. Cependant, l'absence de réservation de bande passante peut entraîner des problèmes de congestion.

On peut aussi prendre le temps de transmission comme point de comparaison. Sur un réseau à commutation de circuits, le temps de transit de l'information est connu. Il dépend uniquement des caractéristiques du circuit. Sur un réseau à commutation de paquets, chaque paquet peut emprunter un itinéraire propre et tous ces itinéraires ne possèdent pas les mêmes caractéristiques. De plus, chaque élément d'interconnexion doit stocker les paquets avant de prendre une décision d'acheminement ; ce qui introduit un temps de latence supplémentaire.

Les débits réseau et la capacité de traitement des équipements d'interconnexion se sont considérablement développés durant les dernières années. De ce fait, les temps de transmission de l'information deviennent négligeables devant d'autres paramètres comme le temps d'accès aux médias de stockage. C'est un des facteurs qui facilite le déploiement de presque tous les nouveaux services sur Internet.

2.5 Services avec et sans connexion

Les services fournis par les couches paires entre deux hôtes d'un réseau peuvent fonctionner selon deux modes principaux : avec et sans connexion.

2.5.a Service orienté connexion

Les communications téléphoniques sont un exemple caractéristique de service orienté connexion. Ce n'est qu'après avoir composé le numéro du correspondant et que celui-ci ait décroché que la conversation peut commencer. De la même façon, il faut que les deux correspondants aient raccroché pour qu'une nouvelle communication puisse être initiée.

Pour généraliser, on parle de trois phases : établissement, maintien et libération de la connexion.

Pendant la phase de maintien, les services des couches paires utilisent un circuit de communication unique physique ou virtuel de bout en bout.

Certains services utilisent la phase d'établissement pour négocier des options d'utilisation du circuit utilisé en phase de maintien. C'est notamment le cas de la couche transport du modèle TCP/IP avec le protocole TCP.

Les opérateurs téléphoniques utilisent beaucoup les services en mode connecté pour préserver la tarification basée sur les temps de communication. En effet, dès que l'information ne circule plus sur un circuit unique, il est beaucoup moins facile de comptabiliser le temps d'une connexion.

2.5.b Service non orienté connexion

Les réseaux locaux privés sont généralement les exemples caractéristiques de services sans connexion. Sur ces réseaux, seuls les volumes d'informations comptent. Si on ne s'intéresse qu'à la quantité d'informations transitant en un point donné, peu importe le chemin emprunté par ces informations.

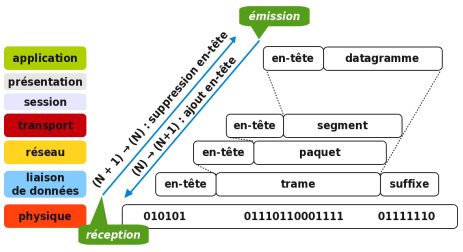
Par opposition à l'analogie avec les communications téléphoniques, on peut utiliser l'exemple du courrier postal. Deux lettres à destination d'une même adresse sont routées indépendamment et peuvent très bien ne pas parvenir dans l'ordre dans lequel elles ont été émises.

Le service de couche réseau de l'Internet avec son protocole IP fonctionne en mode non connecté. Tous les paquets IP sont routés indépendamment et peuvent suivre des chemins différents s'il existe plusieurs itinéraires disponibles.

2.6 Encapsulation

On peut prendre l'exemple des étapes de l'acheminement du courrier postal pour illustrer le concept d'encapsulation. Une lettre est d'abord insérée dans une enveloppe pour être postée. Cette enveloppe est ensuite placée dans un sac postal. Le sac postal est lui-même transporté dans un conteneur. Ces étapes illustrent l'encapsulation de l'information lors de son émission. Pour la réception, on reprend les mêmes étapes dans l'ordre inverse. Le sac postal est extrait du conteneur. L'enveloppe est extraite du sac postal et déposée dans la boîte aux lettres du destinataire. On peut parler de désencapsulation pour décrire les étapes de réception.

Si on aborde le concept de façon plus formelle, on reprend les mêmes étapes entre les couches de la modélisation. Au passage d'une couche N vers la couche inférieure (N-1), le flot de données est enrichi de champs supplémentaires placés en début et/ou en fin. Dans le premier cas, il s'agit d'un en-tête ou préfixe (header) ; dans le second, d'un suffixe (trailer). Ces informations apportées renseignent l'unité de donnée au niveau de la couche qui les a émises (ici N). Ces champs servent donc, lors de la réception par la couche de même niveau (N) de la station destinataire, au traitement que celle-ci doit effectuer. On peut y trouver les adresses source et destination (de niveau N), un contrôle de parité, la longueur concernant le paquet, des bits de priorité, l'identification du protocole de niveau supérieur (N+1) pour le décodage, des numéros d'acquittement, etc.



2.7 Modèles de référence

L'histoire des modélisations réseau a plus de quarante ans. Pour une technologie aussi évolutive que

l'interconnexion des réseaux de télécommunications, quarante ans représentent une durée très longue au cours de laquelle plusieurs révolutions se sont produites.

À l'époque de la suprématie d'IBM, les plus hautes autorités politiques ont craint que cette société n'exerce une mainmise irréversible sur les réseaux en imposant son modèle en 7 couches SNA (lien 105). C'est en partie pour « répondre » à cette crainte que l'ISO (lien 106) a lancé les travaux sur la modélisation OSI (lien 107) : un autre modèle à 7 couches.

Parallèlement, le Département de la Défense des États-Unis a lancé le projet ARPANET (lien 108) : l'ancêtre de l'Internet. Ce réseau qui a relié plus d'une centaine d'universités, a généré sa propre modélisation baptisée du nom de ses deux protocoles phares : TCP et IP. Les profits générés par ce réseau de recherche ont été suffisants pour que les constructeurs investissent dans la conception d'équipements utilisant ses protocoles. Comme ARPANET est devenu Internet et que les profits réalisés sur cette base ont explosé, TCP/IP se retrouve en tête des modèles de référence.

Il faut aussi compter avec l'ITU (lien 109). Cet organisme regroupe les agences réglementaires qui coordonnent la gestion des télécommunications. Parmi ces organismes, on trouve la Federal Communications Commission (FCC) aux USA, l'ARCEP (lien 110) en France. Depuis 1975, le développement des réseaux téléphoniques à commutation de paquets à partir des normalisations X.25 a contribué à figer la structure des couches basses de l'ensemble des modélisations.

3 Modélisation OSI

Il s'agit d'un modèle en 7 couches dont le principe fondamental est de définir ce que chaque couche doit faire mais pas comment elle doit le faire.

Les protocoles et les normalisations IEEE (lien 111) sont là pour définir comment les services sont fournis entre les couches.

3.1 Point fort : la transmission de l'information

Il a fallu quelques années entre la proposition initiale de l'ISO (1978) et la publication du standard IS7498 :84 (1984). C'est ce standard, largement adopté par les constructeurs les plus importants et d'autres organismes de normalisation comme l'ITU, qui a contribué à la popularité de ce « modèle d'interconnexion des systèmes ouverts ». Un temps d'élaboration aussi long donne la mesure de la tâche accomplie pour fédérer les standards existants et obtenir un consensus entre des partenaires aux conceptions très éloignées sur l'organisation des réseaux.

Les spécifications des services des couches dédiées à la transmission de l'information, aussi appelées couches basses, sont si « chargées » qu'il a fallu les subdiviser en sous-couches respectant les modes de fonctionnement avec et sans connexion. Le consensus obtenu si difficilement est encore en vigueur aujourd'hui ; tous les constructeurs d'équipements le respectent.

3.2 Point faible : le traitement de l'information

Alors que pour les couches liées à la transmission de l'information les standards existants ont pesé très lourd dans l'élaboration du modèle, la voie était quasiment libre pour les couches dédiées au traitement de l'information dans les années 80.

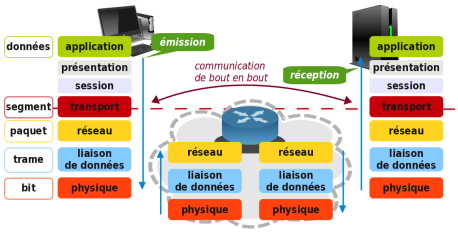
La tâche était, a priori, plus facile sachant qu'il n'y avait pas « d'ordre établi » sur les applications de traitement. Ce sont pourtant ces couches hautes qui ont provoqué le déclin de la modélisation OSI.

Les spécifications qui ont été produites pour les couches transport, session et application étaient si complexes que seules de rares applications mastodontes ont pu être développées. Ces services réseau, tels que les annuaires de services X500 normalisés par l'ISO et l'ITU n'ont pu être popularisés qu'après avoir été simplifiés et allégés. Aujourd'hui, on ne parle plus que des annuaires LDAP (Lightweight Directory Access Protocol : lien 112).

Cette expérience a montré que l'on ne peut pas produire des spécifications trop « distantes » de l'état de la technologie et des infrastructures existantes. L'ISO a renoncé à promouvoir le modèle OSI en décembre 1994 après avoir publié le standard ISO/IEC 7498-1 :1994. À cette époque, l'Internet universitaire était déjà une réalité et les investissements industriels sur TCP/IP étaient bien engagés.

Le plus curieux, c'est que tous les organismes ne semblent pas avoir retenu la leçon. Le Forum ATM (Asynchronous Transfer Mode : technologie réseau répandue dans les grandes dorsales des opérateurs d'interconnexion), a produit deux spécifications contradictoires à un an d'intervalle vers le milieu des années 1990. La technologie ATM est aujourd'hui obsolète et remplacée par des dorsales Ethernet.

3.3 Définitions



Usuellement, on distingue les Couches Hautes (de Transport jusqu'à Application) qui ont une fonction de traitement des données indépendante de la technique de connexion entre deux systèmes des Couches Basses (de Physique jusqu'à Réseau) qui ont une fonction de transmission de l'information liée à la technologie de communication.

3.4 Couche physique (bit)

Elle s'occupe de la transmission « brute » des flots de bits sur un circuit de communication sans connaître ni la structure, ni la signification de ces bits. À ce niveau, on s'intéresse à l'amplitude du signal, à la durée d'un bit, à la possibilité de transmettre simultanément dans les deux sens, à l'établissement et la libération du canal de connexion.

3.5 Couche liaison de données (trame)

Elle transforme les flots de bits en lignes de données sans erreur. Pour cela, elle fractionne les données d'entrée de l'émetteur en trames de données. C'est donc à elle de reconnaître les frontières des trames. Cette fonction entraîne la résolution des problèmes de trames endommagées, perdues ou dupliquées. On retrouve ici la fonction de contrôle d'erreur : lien 113.

C'est aussi à ce niveau que l'on peut trouver des mécanismes de régulation pour éviter la saturation du canal de communication par un émetteur unique. C'est la fonction de contrôle de flux : lien 114. Une technique très simple, employée dans les réseaux Ethernet, qui interdit les émissions continues à tous les hôtes du réseau. Une émission ne peut avoir lieu que si tous les hôtes ont eu le temps matériel de détecter que le média partagé est libre.

Si le service le requiert, le récepteur confirme la réception de chaque trame en émettant une trame d'acquittement.

Les réseaux à diffusion utilisent un service ou une sous-couche spécifique pour contrôler que l'accès au média est libre. Dans le cas des réseaux sans fil de types IEEE 802.11 ou Wi-Fi, des trames de gestion indépendantes des informations utilisateur sont échangées entre les équipements pour contrôler l'accès aux canaux hertziens.

3.6 Couche réseau (paquet)

Elle gère le sous-réseau (les couches basses), c'est-à-dire la façon dont les paquets sont acheminés de l'émetteur au récepteur. Elle contrôle la route empruntée par les paquets.

Les stratégies utilisées pour le routage sont très variables. On peut trouver des tables statiques dans les réseaux qui évoluent rarement. Cependant, on utilise généralement des protocoles de routage plus ou moins sophistiqués, dédiés à l'échange d'informations entre les équipements d'interconnexion fournissant un service de niveau réseau.

En plus des itinéraires disponibles, les choix de route se font en fonction du nombre d'équipements d'interconnexion à traverser, du débit disponible, de la charge d'un lien ou encore du temps de transit entre deux extrémités.

Cette couche doit aussi résoudre les problèmes d'interconnexion entre réseaux hétérogènes. Si un paquet doit transiter par deux réseaux utilisant des technologies différentes, la couche réseau doit gérer :

- le changement de formats d'adresses ;
- le redimensionnement des paquets ;
- la mise en conformité entre protocoles différents ;
- la comptabilisation du coût d'acheminement de l'information.

C'est encore à la couche réseau de contrôler la congestion d'un réseau en notifiant les hôtes voisins à l'aide d'informations spécifiques.

3.7 Couche transport (segment)

Sa fonction de base est de traiter les données de la couche session et de les découper au besoin en petites unités. Ces petites unités sont ensuite transmises à la couche réseau tout en s'assurant qu'elles sont parvenues à destination.

- Elle peut multiplier les connexions réseau si la connexion de transport requiert un débit rapide.
- Elle peut multiplexer les connexions de transport si le maintien d'une connexion réseau est coûteux.

Une connexion de transport est un canal point à point délivrant des messages sans erreur dans l'ordre d'émission. Avec la couche transport, on aborde les couches de « bout en bout » (couches hautes), c'est-à-dire que le même programme s'exécute sur l'émetteur et le destinataire en utilisant les messages d'entête et de contrôle. Cette couche doit gérer l'initialisation et la fin des connexions sur le réseau ; ce qui nécessite un mécanisme d'adressage permettant d'identifier le ou les destinataires. La couche transport est parfois considérée comme faisant partie des couches basses parce qu'elle doit préserver la

couche session des changements de technologie entre réseaux.

3.8 Couche session

Elle permet à des utilisateurs, opérant sur différentes machines, d'établir des sessions entre eux. Une session a pour but le transport des données. Par rapport à la couche transport, elle offre des services supplémentaires tels que :

— La gestion du dialogue ou du jeton : certains protocoles utilisent des jetons (autorisation d'émission) que les machines d'un réseau peuvent s'échanger.

— La synchronisation : cette technique consiste à insérer des éléments tests dans le flot de données de manière à ne pas devoir reprendre la totalité d'une opération en cas d'échec.

C'est à travers la couche session qu'un utilisateur peut accéder à un système à temps partagé distant ou transférer des fichiers.

3.9 Couche présentation

Elle traite la syntaxe de l'information transmise. Elle assure l'encodage et/ou la compression des données dans une norme agréée.

Elle assure des conversions telles que celles des protocoles d'utilisation de terminaux incompatibles entre eux, celles entre les différents systèmes de fichiers ou encore celles des formats du courrier électronique.

3.10 Couche application

Cette couche assure l'interface entre l'utilisateur et les services du réseau. On y trouve toutes les applications cliente ou serveur connues : transfert de fichiers, courrier électronique, Web, multimédia, etc.

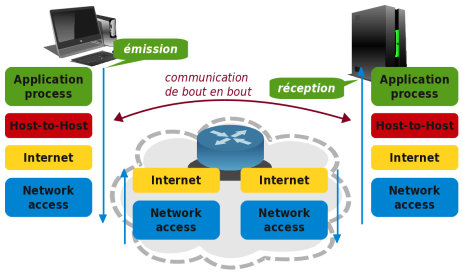
4 Modélisation TCP/IP

Le nom de ce modèle de référence provient de ses deux principaux protocoles. Ce modèle est apparu en 1974 avec la construction de l'ancêtre militaire de l'Internet, l'ARPANET (lien 115). Les objectifs principaux de cette modélisation sont :

— relier des réseaux hétérogènes de façon transparente (lignes téléphoniques, réseaux locaux, etc.) ;

— garantir les connexions quel que soit l'état des lignes de transmission (commutation de paquets) ;

— assurer le fonctionnement d'applications très différentes (transfert de fichier, multimédia, etc.).



— Network Access : la couche d'accès réseau a pour rôle de transmettre les données sur le média physique utilisé. En fonction du type de réseau, des protocoles différents peuvent être utilisés à ce niveau.

— Internet : la couche interréseaux a pour rôle de transmettre les données à travers une série de réseaux physiques différents qui relient un hôte source avec un hôte destination. Les protocoles de routage sont étroitement associés à ce niveau. IP est le protocole routé de base sur l'Internet.

— Host-to-Host : La couche hôte-à-hôte prend en charge la gestion de connexion, le contrôle de flux, la retransmission des données perdues et d'autres modes de gestion des flux. Les protocoles TCP et UDP sont dédiés à ces fonctions de transport.

— Process/Application : La couche application sert à l'exécution des protocoles de niveau utilisateur tels que les échanges de courrier électronique (SMTP), le transfert de fichiers (FTP) ou les connexions distantes (telnet).

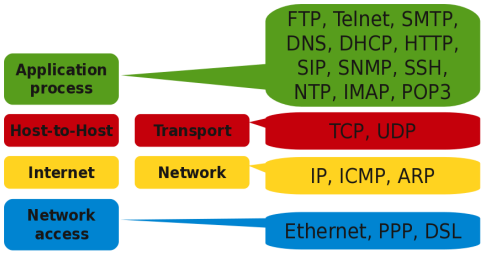
4.1 Point fort : les protocoles

Le fait que ce modèle porte le nom de ces protocoles est lourd de signification. Si la démarche de recherche de consensus dans le développement du modèle s'apparente à la démarche suivie pour le modèle OSI (lien 116), les spécifications ont été directement accessibles pour un public beaucoup plus large.

C'est ce principe de publication de RFC (Request for comments : lien 117) qui a favorisé le développement des protocoles au profit du modèle. Tous les protocoles de l'Internet ont été « standardisés » à l'aide de ces documents. Lorsque quelqu'un met au point un protocole, il le soumet à la communauté à l'aide d'un document RFC. Ce travail est ensuite repris et amélioré par d'autres qui publient un nouveau RFC et ainsi de suite. C'est la démarche d'origine de développement des logiciels libres.

Ce travail à base de propositions ouvertes s'est montré très efficace puisqu'il a supplanté le modèle

issu de l'ISO, l'organisme officiel de normalisation. Dès les premiers documents RFC, les piles de protocoles (lien 118) ont été illustrées.



4.2 Point faible : le modèle

La notion de modélisation n'est pas apparue comme une priorité relativement au développement des protocoles. Pour répondre aux objectifs du modèle Internet, ses développeurs ne se sont que très peu intéressés aux modes de transmission de l'information. Ils devaient utiliser l'existant de façon transparente. Le modèle Internet est donc très incomplet sur les aspects transmission.

En reprenant le principe à l'origine du réseau de communication militaire ARPANET (lien 119), on doit pouvoir communiquer d'un point à un autre de l'Internet « quel que soit l'état du réseau ». Une grande partie de l'infrastructure peut être détruite par une frappe nucléaire et les communications doivent toujours être possibles. C'est ce mode de fonctionnement singulier qui a conduit à l'adoption d'un réseau à commutation de paquets fonctionnant en mode non connecté sans aucune hiérarchie. Si un autre principe avec supervision, hiérarchie et mode connecté avait été retenu, il suffirait qu'un point névralgique soit touché pour interrompre l'ensemble des communications.

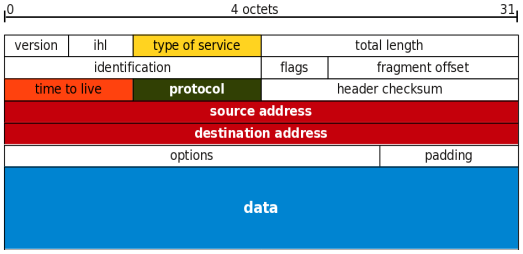
C'est donc sur les couches basses qui traitent de la transmission de l'information que le modèle OSI conserve l'avantage. Le protocole IP est une implémentation particulière des fonctions de la couche réseau décrites dans le modèle OSI. De la même façon, les protocoles TCP et UDP sont des implémentations particulières des fonctions de la couche transport.

4.3 Couche Internet : le protocole IP

Pour répondre aux objectifs énoncés ci-dessus, le principe d'un réseau à commutation de paquets en mode non connecté a été retenu. Ce type de réseau correspond à un mode particulier d'utilisation de la couche réseau (3) du modèle OSI (lien 120).

Dans cette organisation, le rôle de la couche Internet est de transmettre des paquets sur n'importe quel type de liaison indépendamment les uns des autres. Les paquets émis dans un certain ordre peuvent ainsi être reçus dans un autre ordre en différents fragments.

Le fonctionnement de la couche réseau du modèle TCP/IP est décrit dans le document standard RFC791 Internet Protocol (lien 121). L'absence totale de mécanisme de contrôle et de correction d'erreur est une caractéristique importante qui découle des mêmes principes. La fiabilisation des communications ne se joue pas au niveau réseau mais au niveau transport.



Version : 4 bits : version du protocole IP codée sur 4 bits : 0100 pour IPv4 et 0110 pour IPv6.

Internet Header Length : 4 bits, IHL : longueur de l'en-tête en mots de 32 bits. Cette valeur est utilisée pour distinguer la partie en-tête de la partie données du paquet. La représentation usuelle de l'en-tête se fait sur 32 bits de largeur. Comme les champs Options et Padding ne sont pas obligatoires, la valeur minimum du champ IHL est 5 (0101).

Type Of Service : 8 bits, TOS : champ découpé en deux parties. Les trois premiers bits sont appelés precedence et les cinq derniers représentent le type de service. La définition d'origine prévoyait trois choix : low-delay, high-reliability et high-throughput. Ce « marquage » des paquets est utilisable pour définir des flux prioritaires sur une interconnexion réseau « sous contrôle ». Sur l'Internet, les opérateurs définissent leurs propres priorités ; donc leurs propres valeurs pour ce champ. Voir le document HOWTO du routage avancé et du contrôle de trafic sous Linux : lien 122.

Total Length : 16 bits : longueur du datagramme : en-tête et données. La taille minimum est de 21 octets (en-tête + 1 octet de données). Comme ce champ est représenté sur 16 bits, la taille maximum est de 2Î6 - 1, soit 64 Ko.

Identification : 16 bits : chaque paquet IP reçoit un numéro d'identification à sa création. Il est possible qu'un paquet soit découpé en fragments avant d'atteindre sa destination finale. Chaque fragment appartient au même paquet IP. Chaque fragment possède le même numéro d'identification.

Flags : 3 bits, ce champ contient trois indicateurs d'état :

- Reserved flag : doit toujours être à 0 ;
- Don't Fragment (DF) : à 0 si le paquet peut être fragmenté ; à 1 s'il ne doit pas être fragmenté ;
- More Fragments (MF) : à 1 si d'autres fragments sont attendus ; à 0 s'il n'y a pas/plus de fragments.

Fragment Offset : 13 bits : position du fragment dans le datagramme courant. Cette position est comptée en octets.

Time To Live : 8 bits, TTL : ce compteur est décrémenté à chaque traversée de routeur. Si la valeur 0 est atteinte, le paquet est jeté. Cela signifie qu'il ne peut être délivré à sa destination finale. La valeur initiale du champ TTL dépend du système d'exploitation utilisé.

Protocol : 8 bits : ce champ spécifie le protocole utilisé dans les données du paquet IP. Par exemple, la valeur 1 indique que le protocole utilisé est ICMP. On sait ainsi que ce paquet n'est pas destiné à une application. Les différentes valeurs de ce champ sont listées dans le fichier /etc/protocols sur les systèmes GNU/Linux ou *BSD.

Header Checksum : 16 bits : à chaque création ou modification d'un paquet, une somme de contrôle (cyclic redundancy check) est calculée sur son en-tête. Lorsque le paquet arrive à destination, cette somme est recalculée. Si le résultat diffère, c'est que le paquet a été endommagé lors de son trajet.

Source Address : 32 bits : adresse IP de l'hôte qui a émis le paquet.

Destination Address : 32 bits : adresse IP de l'hôte qui doit recevoir le paquet.

Options and Padding : cette partie de l'en-tête est optionnelle. Ce champ est utilisé pour fournir des instructions spécifiques de distribution du paquet qui ne sont pas couvertes par les autres champs de l'en-tête. La taille maximum de ces instructions est limitée à 40 octets regroupés en doubles mots de 32 bits. Les bits de padding servent à compléter le dernier double mot de 32 bits.

Data : c'est le dernier champ du paquet IP. Il contient les « données » du paquet. Celles-ci peuvent débuter par un en-tête de couche transport (4) qui donnera d'autres instructions à l'application qui recevra les données. Le champ Data peut aussi contenir un message ICMP qui ne contient aucune donnée utilisateur.

Le document Adressage IPv4 (lien 123) présente le format des adresses du protocole IP ainsi que les différentes évolutions des mécanismes de découpage du plan d'adressage en groupes logiques adaptés aux différents usages.

4.4 Couche Host-to-Host : les protocoles TCP & UDP

Avec la couche transport, on aborde le domaine des communications de bout en bout indépendantes de l'état du sous-réseau. Les paquets de la couche réseau peuvent être acheminés à destination par des chemins différents et dans le désordre. Par nature, le protocole IP n'offre pas de garantie. Les routeurs peuvent se débarrasser des paquets suivant plusieurs critères tels que des erreurs sur les sommes de contrôle, des congestions de trafic sur les interfaces réseau ou des adresses ne correspondant à aucune route connue. Tous les programmes et services de la couche application ne peuvent se contenter d'un mode de fonctionnement aussi « fragile ». C'est une des raisons pour laquelle deux protocoles distincts ont été développés pour la couche transport.

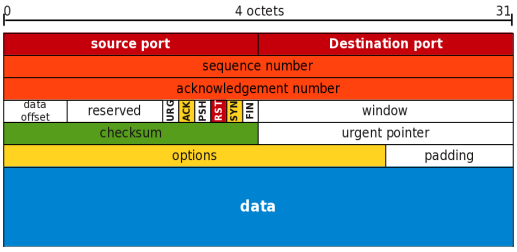
4.5 Protocole TCP

Historiquement, TCP est le premier protocole de transport développé pour l'Internet. Les premières spécifications ARPANET (lien 124) prévoyaient un transport de l'information très fiable indépendant du type et de l'état du réseau. Le fonctionnement du protocole TCP a été décrit dans le document RFC793 Transmission Control Protocol : lien 125.

- **Protocole de bout en bout.** Les processus pairs des couches transport de deux équipements connectés dialoguent l'un avec l'autre sans rien connaître du réseau sous-jacent. Les numéros de port source et destination présents dans l'en-tête de segment servent à adresser les processus de couche application en communication.
- **Protocole orienté connexion.** La fiabilité du transport TCP dépend de l'établissement d'une connexion entre les processus pairs qui veulent dialoguer. L'établissement d'une connexion est réalisé par l'échange d'informations telles que les numéros de ports, les numéros de séquence et la taille de fenêtre.
- **Multiplexage à l'aide des numéros de ports.** Les numéros de ports constituent le mécanisme d'adressage de la couche transport. Ils servent à désigner le processus de la couche application utilisé pour l'émission ainsi que celui utilisé pour la réception.
- **Transfert de données segmenté et ordonné.** Le flux des données issues de la couche application est segmenté et comptabilisé lors de l'encapsulation puis délivré dans le même ordre au processus qui le reçoit.
- **Récupération sur erreur.** L'utilisation des numéros de séquence et d'acquittement permet de comptabiliser les données transmises et de reprendre l'émission des données non reçues.
- **Contrôle de flux avec fenêtrage.** La combinaison de l'utilisation des numéros de séquence et d'acquittement avec la notion de fenêtre permet de contrôler la quantité de données à transmettre avant de procéder à un acquittement. Au début des échanges, la taille de fenêtre est réduite. Si aucune erreur ne survient, cette taille de fenêtre augmente suivant une règle définie. Au contraire, si des erreurs surviennent, la taille de fenêtre diminue de façon à augmenter le nombre des contrôles.

4.5.a En-tête TCP

Les fonctions d'établissement, de maintien, de libération et de contrôle des échanges ont conduit au développement d'un en-tête comprenant un grand nombre de champs.



Source Port : 16 bits : numéro du port source. Ce numéro correspond au point de communication (socket inet) utilisé par le service de la couche application de l'émetteur.

Destination Port : 16 bits : numéro du port destination. Ce numéro correspond au point de communication (socket inet) utilisé par le service de la couche application du destinataire.

Sequence Number : 32 bits

- Le protocole TCP a besoin de garder une trace de toutes les données qu'il reçoit de la couche application de façon à être sûr qu'elles ont bien été reçues par le destinataire. De plus, le protocole doit être sûr que ces données ont été reçues dans l'ordre dans lequel elles ont été envoyées. Il doit retransmettre toute donnée perdue.
- On affecte un numéro de séquence à chaque octet de données pour en garder une trace lors du processus de transmission, réception et acquittement. Dans la pratique, ce sont des blocs d'octets qui sont gérés en utilisant les numéros de séquence de début et de fin de bloc.
- Les numéros de séquence sont nécessaires à la mise en oeuvre du système de fenêtre glissante du protocole TCP. C'est ce système qui garantit fiabilité et contrôle de flots de données.

Acknowledgment Number : 32 bits : le rôle des numéros d'acquittement est le même que celui des numéros de séquence. Simplement, chaque extrémité en communication initie son propre jeu de numéros. Ainsi chaque extrémité assure la fiabilisation et le contrôle de flux de façon autonome.

Data Offset : 4 bits : nombre de mots de 32 bits contenus dans l'en-tête TCP. Indication du début des données. Tout en-tête TCP, avec ou sans options, est un multiple de mots de 32 bits.

Reserved : 6 bits : champ réservé pour une utilisation ultérieure. Les 6 bits doivent être à 0.

Control bits : 6 bits : ces bits sont les indicateurs d'état qui servent à l'établissement, au maintien et à la libération des connexions TCP. Leur rôle est essentiel dans le fonctionnement du protocole.

- URG : indique que le champ Urgent Pointer est significatif. Une partie des données du segment sont urgentes.
- ACK : indique que le champ Acknowledgment field est significatif. Le segment acquitte la transmission d'un bloc d'octets.
- PSH : indique à l'hôte en réception de « pousser » toutes les informations en mémoire tampon vers l'application en couche supérieure. L'émetteur notifie le récepteur qu'il a transmis toutes ses données « pour l'instant ».
- RST : indique un arrêt ou un refus de connexion.
- SYN : indique une demande de synchronisation de numéro de séquence. Demande d'ouverture de connexion TCP.
- FIN : indique que l'émetteur n'a plus de données à transmettre. Demande de libération de connexion.

Window : 16 bits : nombre d'octets de données à transmettre à partir de celui indiqué par le champ Acknowledgment.

Checksum : 16 bits : somme de contrôle sur 16 bits de l'en-tête et des données.

Urgent Pointer : 16 bits : ce champ est interprété uniquement si le bit de contrôle URG est à 1. Le pointeur donne le numéro de séquence de l'octet qui suit les données urgentes.

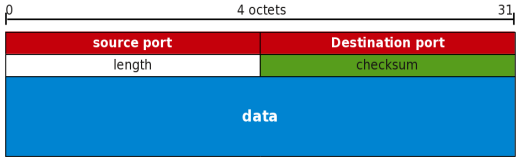
Options : variable entre 0 et 44 octets : il existe deux formats d'options : un seul octet de catégorie d'option ou un octet de catégorie d'option suivi d'un octet de longueur d'option et de l'octet des données de l'option.

4.6 Protocole UDP

Le protocole UDP est apparu avec le développement des réseaux locaux dont la fiabilité est connue à priori. Il permet de s'affranchir des fonctions de contrôle. C'est un protocole minimum sans garantie de délivrance des messages et sans séquencement. En conséquence, l'en-tête est très nettement simplifié et le nombre de champs est très réduit.

Ce protocole présente un grand intérêt dans les applications orientées temps réel dans la mesure où il n'introduit aucune latence relativement aux fonctions de contrôle de flux de TCP.

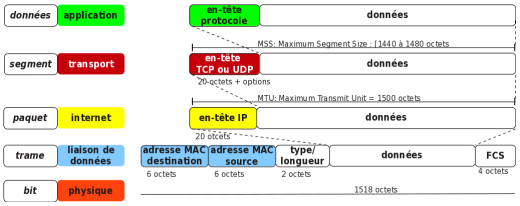
4.6.a En-tête UDP



Les numéros de ports constituent le mécanisme d'adressage pour les communications de bout en bout comme dans le cas du protocole TCP.

4.7 Unités de données

Les protocoles listés ci-avant échangent des données entre eux lors du passage d'une couche à l'autre. On parle d'unité de données de protocole ou Protocol Data Unit (PDU). Cette notion de PDU est générale et n'est pas très employée en dehors des présentations sur les modélisations et l'étude du protocole Spanning Tree au niveau liaison de données. Avec l'utilisation systématique des protocoles de l'Internet, on a introduit un vocabulaire spécifique à chaque couche. Voici un schéma sur lequel figure ce vocabulaire ainsi que les dimensions en octets de chaque élément.



Cette représentation fait apparaître le format de trame Ethernet. Même si la technologie Ethernet n'est pas directement liée aux protocoles de l'Internet, son format de trame tend à devenir universel. On le retrouve avec les technologies Wi-Fi, les connexions ADSL/PPPOE et même de plus en plus sur les réseaux étendus.

Avant 1997, date à laquelle l'IEEE (lien 126) a incorporé le format « historique » de trame Ethernet dans le standard officiel, on devait systématiquement distinguer deux formats de trames suivant le champ Type/Longueur.

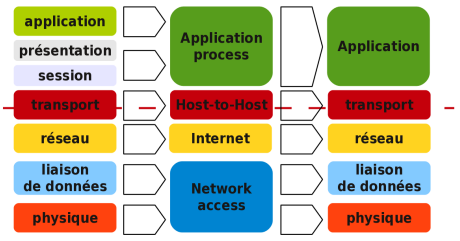
- La définition de trame Ethernet II, celle qui utilise le champ type, a été intégrée avec les protocoles de l'Internet à partir des documents RFC894 Standard for the transmission of IP datagrams over Ethernet networks (lien 127) et RFC1042 Standard for the transmission of IP datagrams over IEEE 802 networks (lien 128). Le champ type de la trame indique le type du protocole de couche supérieure ; IP ou ARP dans la plupart des cas. Pour plus d'informations, voir la référence sur les ETHER TYPES : lien 129.
- La définition de trame IEEE 802.3 initiale, celle qui utilise le champ longueur, n'est jamais utilisée pour le trafic IP ; donc pour le trafic utilisateur. Seules les communications spécifiques entre équipements réseau utilisent ce format de trame associé à des protocoles spécifiques.

5 Modèle Contemporain

Cette dénomination n'a vraiment rien d'officiel. Elle signifie simplement qu'actuellement la conception et l'exploitation des réseaux se fait à partir de la synthèse entre le modèle OSI (lien 130) et le modèle Internet (lien 131).

Chaque modèle a compensé les faiblesses de l'autre.

- **Le modèle OSI a structuré les relations entre les couches basses.** Sans ce travail de normalisation, l'évolution des réseaux de télécommunications était compromise. L'interconnexion entre systèmes propriétaires hétérogènes aurait toujours posé problème.
- **Le modèle Internet a structuré les protocoles d'applications par services.** C'est le développement libre des services (noms de domaine, courrier, Web) au-dessus des couches basses du réseau en tenant compte des infrastructures existantes qui a permis l'explosion de l'Internet là où les spécifications trop lourdes ont complètement échoué.



On peut aussi illustrer la comparaison entre les deux modèles en partant du positionnement des protocoles TCP (RFC793 Transmission Control Protocol : Transmission Control Protocol (lien 132)) et IP (RFC791 Internet Protocol : Internet Protocol (lien 133)) qui datent de 1981.

```
1  Couches| Couches     | RFC 791                                    |Système
2  OSI    | TCP/IP      |                                            |d'exploitation
3  -------+-------------+--------------------------------------------+--------------
4   7     |             |+------+ +-----+ +-----+      +-----+      |Processus
5   6     | Application||Telnet| | FTP | | TFTP| ... | ... |      |utilisateur
6   5     |             |+------+ +-----+ +-----+      +-----+      |'userspace'
7         |.............|       |   |         |          |          |..............
8         |             |      +-----+      +-----+      +-----+    |Code du noyau
9   4     | Transport   |      | TCP |      | UDP | ... | ... |    |'kernelspace'
10        |             |      +-----+      +-----+      +-----+    |
```

```
11      |.............|         |           |           |       | . . . . . . . .
12      |             |      +--------------------------+----+  |Adressage
13    3 | Réseau      |      |    Internet Protocol & ICMP    |  |Logique
14      |             |      +--------------------------+----+  |
15      |.............|                      |                    | . . . . . . . .
16    2 |             |        +---------------------------+      |Adressage
17    1 | Accès       |        |   Local Network Protocol  |      |Physique
18      |             |        +---------------------------+      |
```

6 En guise de conclusion

Une fois les concepts des modélisations réseau introduits, une bonne compréhension du fonctionnement de chaque couche suppose que l'on se penche sur les technologies et les protocoles correspondants. En remontant les niveaux du modèle contemporain proposé ci-avant, voici la liste des articles et des supports de cours à consulter, suivie d'un glossaire sur les acronymes couramment utilisés.

6.1 Documents de référence

— Technologie Ethernet : la technologie Ethernet et les normalisations IEEE 802.3 dominent très largement au niveau physique (couche 1) et liaison (couche 2) dans le monde des réseaux locaux : lien 134 ;

— Adressage IPv4 : l'adressage IPv4 synthétise les caractéristiques de fonctionnement du protocole de niveau réseau (couche 3) du modèle TCP/IP : lien 135 ;

— Analyse réseau : toutes les descriptions académiques « livresques » ne valent pas grand-chose sans illustration à partir du trafic réseau réel. L'analyse réseau est l'outil pédagogique fondamental pour s'assurer d'une bonne compréhension des mécanismes de fonctionnement des protocoles réseau. Voir Introduction à l'analyse réseau avec Wireshark : lien 136

6.2 Glossaire des acronymes

ARCEP : Autorité de Régulation des Communications Électroniques et des Postes (lien 137) ARPANET : Advanced Research Projects Agency NETwork

— Ancêtre militaire de l'Internet développé aux États-Unis à partir des années 60.

IEEE : Institute of Electrical and Electronics Engineers (lien 138)

— Organisme responsable de la publication des normes sur les technologies réseaux, notamment Ethernet.

ISO : International Standard Organisation (lien 139) :International Organization for Standardization

— Organisme international responsable de la publication de nombreuses normes dont la modélisation OSI.

ITU : International Telecommunication Union (lien 140)

— Organisme international responsable de la publication des normes sur les télécommunications.

OSI : Open Systems Interconnection

— Le Modèle de référence de base pour l'interconnexion de systèmes ouverts (lien 141) est une norme (ISO 7498) composée de 4 parties

SNA : Systems Network Architecture

— Modélisation propriétaire promue par IBM dans les années 70 et aujourd'hui abandonnée.

Retrouvez l'article de ***Philippe Latu*** *en ligne : lien 142*

OpenOffice-LibreOffice

Les derniers tutoriels et articles

Apprendre à créer un tableau croisé dynamique avec LibreOffice Calc

Voici un petit tutoriel pour apprendre à créer un tableau croisé dynamique avec LibreOffice Calc.

1 Introduction

Le tableau croisé dynamique (TCD), encore appelé rapport de tableau croisé dynamique, est un excellent outil d'analyse et de synthèse de données. En effet, il permet de compiler, de regrouper et d'analyser des informations brutes préalablement recueillies. Le tableau est dynamique ; cela veut dire qu'il est possible d'y ajouter ou retirer des données et de modifier sa structure. Toute modification apportée sur les données sources est automatiquement prise en compte au niveau TCD après actualisation de ce dernier.

Le TCD, qui porte le nom de « table pilote » dans LibreOffice, est donc en mesure de traiter une multitude d'informations et sa présentation est fonction :

- des champs ou colonnes à analyser ;
- des regroupements à faire ;
- et des formules prédéfinies à utiliser.

Dans ce tutoriel, je vais vous faire découvrir les différentes étapes de la création d'un TCD afin de vous permettre de réaliser les vôtres. Toutefois, il est essentiel de savoir ce que l'on veut obtenir avant de se lancer dans la création du TCD.

2 Prérequis

Pour réaliser un tableau croisé dynamique sous LibreOffice Calc, il est impératif de définir la source de données. Cette dernière peut se faire suivant trois méthodes :

- soit à partir de la sélection active contenue dans la feuille de calcul utilisée ;
- soit via les plages nommées ;
- soit à partir d'une base de données externe qui aura été liée à LibreOffice.

Le tableau dans lequel les données sources sont enregistrées de façon brute pourrait être considéré comme une base de données, car les informations qui y sont consignées devront respecter un certain nombre de règles, notamment :

- un titre pour chaque champ ou colonne. Les titres doivent être clairs et précis, car ils représenteront des identifiants dans le tableau croisé dynamique ;
- les données d'un même champ doivent être de même nature ;
- et les champs ou colonnes ne doivent contenir ni sous-totaux, ni filtres.

Dans ce tutoriel, les données sources servant de base à l'élaboration du TCD sont consignées dans une feuille de calcul.

3 Les différentes étapes de la création d'un TCD

La création d'un tableau croisé dynamique est rendue possible grâce à un assistant. Pour ce faire, il faut, dans notre cas, sélectionner le tableau source avant d'exécuter l'assistant prévu à cet effet. Il est aussi possible de procéder à cette sélection sans redéfinir les plages si, au préalable, les cellules concernées

avaient été nommées. Il convient de préciser que le tableau source doit être intégralement sélectionné, y compris les titres des colonnes.

Pour illustrer mes explications, je partirai de l'exemple ci-après.

Exemple :

soit la société X spécialisée dans la téléphonie. Durant le premier trimestre 2013, cette société a vendu des cartes de recharges à ses clients et les produits tirés de ses ventes sont récapitulés dans le tableau ci-après :

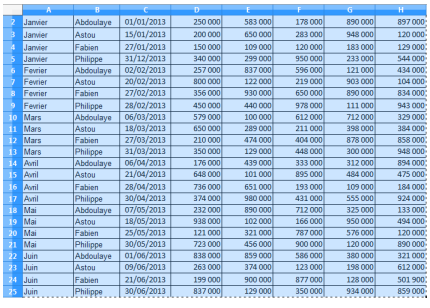
Mois	Clients	Dates	R1.000	R2.000	R5.000	R10.000	R25.000
Janvier	Abdoulaye	01/01/2013	250 000	583 000	178 000	890 000	897 000
Janvier	Astou	15/01/2013	200 000	650 000	283 000	948 000	120 000
Janvier	Fabien	27/01/2013	150 000	109 000	120 000	183 000	129 000
Janvier	Philippe	31/12/2013	340 000	299 000	950 000	233 000	544 000
Fevrier	Abdoulaye	02/02/2013	257 000	837 000	596 000	121 000	434 000
Fevrier	Astou	20/02/2013	800 000	122 000	219 000	903 000	104 000
Fevrier	Fabien	27/02/2013	356 000	930 000	650 000	890 000	834 000
Fevrier	Philippe	28/02/2013	450 000	440 000	978 000	111 000	943 000
Mars	Abdoulaye	06/03/2013	579 000	100 000	612 000	712 000	329 000
Mars	Astou	18/03/2013	650 000	289 000	211 000	398 000	384 000
Mars	Fabien	27/03/2013	210 000	474 000	404 000	878 000	858 000
Mars	Philippe	31/03/2013	350 000	129 000	448 000	300 000	948 000
Avril	Abdoulaye	06/04/2013	176 000	439 000	333 000	312 000	894 000
Avril	Astou	21/04/2013	648 000	101 000	895 000	484 000	475 000
Avril	Fabien	28/04/2013	736 000	651 000	193 000	109 000	184 000
Avril	Philippe	30/04/2013	374 000	980 000	431 000	555 000	924 000
Mai	Abdoulaye	07/05/2013	232 000	890 000	712 000	325 000	133 000
Mai	Astou	18/05/2013	938 000	102 000	166 000	950 000	494 000
Mai	Fabien	25/05/2013	121 000	321 000	787 000	576 000	120 000
Mai	Philippe	30/05/2013	723 000	456 000	900 000	120 000	890 000
Juin	Abdoulaye	01/06/2013	838 000	859 000	586 000	380 000	321 000
Juin	Astou	09/06/2013	263 000	374 000	123 000	198 000	612 000
Juin	Fabien	21/06/2013	199 000	900 000	877 000	128 000	501 900
Juin	Philippe	30/06/2013	837 000	129 000	350 000	934 000	859 000

— R1.000 désigne les recharges de 1 000 FCFA ;
— R2.000 désigne les recharges de 2 000 FCFA ;
— R5.000 désigne les recharges de 5 000 FCFA ;
— R10.000 désigne les recharges de 10 000 FCFA ;
— et R25.000 désigne les recharges de 25 000 FCFA.

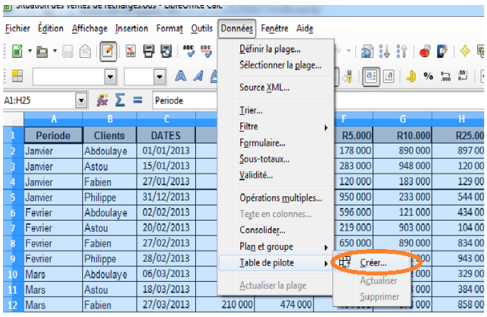
Le PDG de la société souhaite obtenir, sous forme de synthèse, l'état récapitulatif du chiffre d'affaires réalisé avec chaque client et pour chaque type de recharge.

4 Identification des données sources et exécution de l'assistant de création du TCD

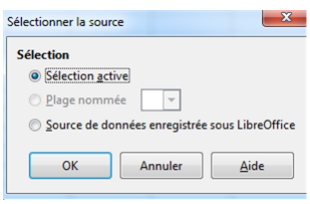
Avant d'exécuter l'assistant, nous sélectionnons d'abord le tableau source des données tout en y incluant les noms des champs. Toutefois, si les cellules ont été nommées, la sélection décrite précédemment devient inutile.



Après avoir sélectionné le tableau source, nous procédons à l'exécution de l'assistant de création d'un tableau croisé dynamique à partir de la barre d'outils de LibreOffice Calc.

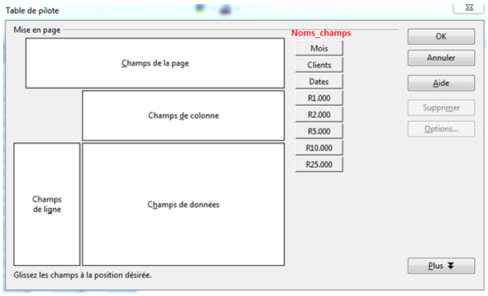


Après cette étape, une fenêtre apparaît, ce qui nous permettra de définir la source des données.



L'exploitation de la fenêtre ci-dessus montre que c'est la sélection précédemment effectuée qui est pro-

posée par défaut. Donc il s'agira pour nous de cliquer directement sur le bouton « **OK**. ». Par la suite, une nouvelle fenêtre permettant de définir la structure de notre TCD s'affiche à l'écran.



L'exploitation du tableau ci-dessus montre que les noms des champs ou colonnes sont affichés sous forme de boutons, c'est pour cela qu'il est important de bien les nommer au départ. Le tableau est aussi composé de quatre zones présentées comme suit :

— la zone « **Champs de la page** » : permettra d'effectuer un filtre du tableau suivant les critères correspondant aux champs insérés dans ladite zone. Toutefois, il est possible d'effectuer des filtres sur les champs figurant dans les autres zones ; Dans notre exemple, lorsque nous plaçons le bouton « mois » dans cette zone et que nous cochons uniquement le mois de janvier, un TCD avec uniquement les ventes réalisées sur le mois de janvier apparaît ;

— la zone « **Champs de colonne** » : cette dernière fera apparaître les titres des colonnes du tableau croisé dynamique et affichera pour colonne donnée les valeurs correspondantes ;

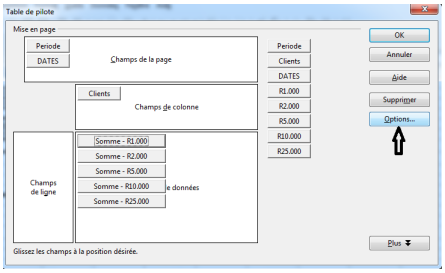
Dans notre exemple, si nous glissons le bouton « Clients » dans cette zone, nous obtenons comme « noms de colonnes » les noms des différents clients de la société, notamment Abdoulaye, Astou, Fabien et Philippe ;

— la zone « **Champs de ligne** » : cette dernière fera apparaître les titres des lignes du tableau croisé dynamique, ces titres correspondront aux différentes valeurs de la colonne du tableau source ;

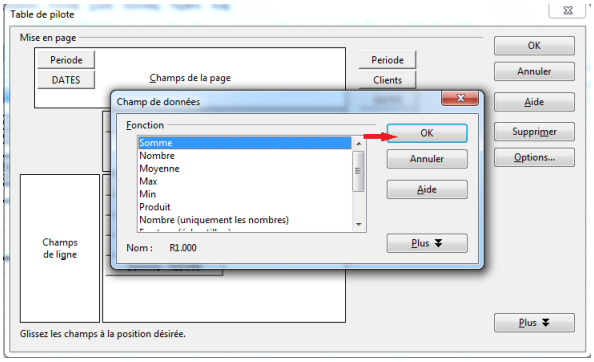
— la zone « **Champs de données** » affichera le résultat attendu. Par défaut, si les données sont composées de chiffres, LibreOffice Calc procédera à une addition ; il est toutefois possible d'utiliser d'autres fonctions. Cela étant, pour modifier une fonction, il faut procéder comme suit :

- cliquer sur chaque champ de la zone « Champs de données »,

- cliquer sur le bouton « **Options** » situé à droite de la fenêtre,



- et dans la boite de dialogue qui s'affiche à l'écran, nous choisissons la formule désirée avant de cliquer sur le bouton « **OK** ».



Il convient de préciser que si les données sources sont de type « texte », le tableur affichera le nombre d'enregistrements ayant satisfait aux critères.

Afin de générer l'état souhaité, nous devrons procéder par des glisser-déposer ; en d'autres termes, il s'agira de glisser les champs dans la zone désirée. Dans notre exemple, nous souhaitons obtenir l'état récapitulatif du chiffre d'affaires réalisé avec chaque client et pour chaque type de recharge via l'outil création d'un TCD.

4.1 Création du tableau croisé dynamique

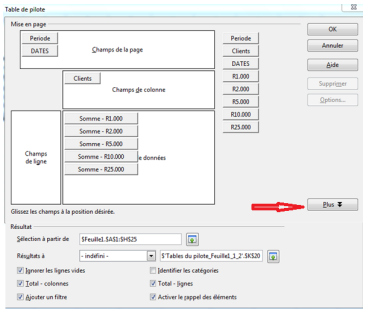
Pour obtenir l'état récapitulatif du chiffre d'affaires cité plus haut, nous procéderons comme suit :

— cliquer successivement sur les boutons « Periode » et « Dates » et les faire glisser dans la zone « Champs de la page ». Cette manipulation permettra de filtrer le résultat obtenu en fonction des périodes ou des dates qu'on aura choisies ;

— cliquer sur le bouton « Clients » et le déposer dans la zone « Champs de colonne » ;

— glisser un à un et successivement les boutons « R1.000 », « R2.000 », « R5.000 », « R10.000 » et « R20.000 » dans la zone « Champs de données ».

Les opérations relatives au glisser-déposer que nous venons d'effectuer sont schématisées comme suit :



La configuration du TCD étant finie, nous faisons un clic sur le bouton « **OK** » et automatiquement le tableur génère un rapport de tableau croisé dynamique présentant la situation du chiffre d'affaires par type de produit et par client sur le premier trimestre de l'année 2013. L'état ainsi obtenu se présente comme suit :

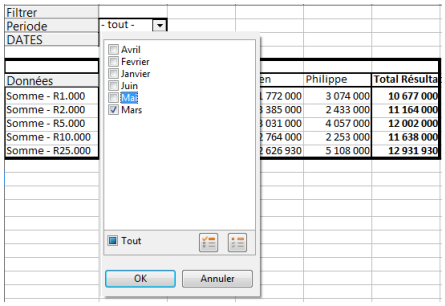
	A	B	C	D	E	F
1	Filtrer					
2	Periode	- tout -				
3	DATES	- tout -				
4						
5		Clients				
6	Données	Abdoulaye	Astou	Fabien	Philippe	Total Résultat
7	Somme - R1.000	2 332 000	3 499 000	1 772 000	3 074 000	**10 677 000**
8	Somme - R2.000	3 708 000	1 638 000	3 385 000	2 433 000	**11 164 000**
9	Somme - R5.000	3 017 000	1 897 000	3 031 000	4 057 000	**12 002 000**
10	Somme - R10.000	2 740 000	3 881 000	2 764 000	2 253 000	**11 638 000**
11	Somme - R25.000	3 008 000	2 189 000	2 626 900	5 108 000	**12 931 900**

4.2 Effectuer des filtres sur les données du tableau croisé dynamique

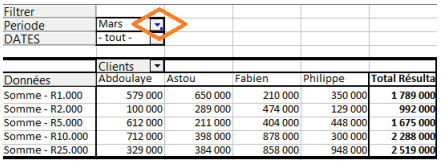
— Filtre sur la période

À partir du résultat obtenu, il est possible de filtrer le tableau croisé dynamique soit par « Periode », soit par « DATES ». En effet, le tableur affiche par défaut le chiffre d'affaires globalement réalisé sur le premier trimestre 2013.

Toutefois, nous avons la possibilité de filtrer les données en fonction de la période souhaitée. Pour illustrer cet état de fait, nous allons filtrer le tableau de façon à ne voir que le chiffre d'affaires réalisé sur le mois de mars. Pour ce faire, à partir du tableau initial, nous faisons un clic sur la flèche se trouvant à côté du champ « Periode ». Sur la liste déroulante qui s'affiche, toutes les cases relatives aux périodes doivent être désactivées à l'exception de celui du mois pour lequel nous souhaitons afficher les informations, dans notre cas le mois de mars.



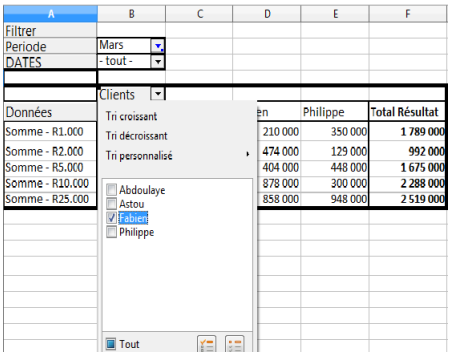
Après un clic sur « **OK** », nous obtenons le résultat ci-après. L'on peut noter que le sélecteur change de couleur avec un point, cela indique qu'un filtre est actif sur ce paramètre (voir image ci-dessous) :



— Filtre sur le Client

Il est aussi possible de filtrer sur le client de façon à ne faire apparaître que le chiffre d'affaires réalisé avec un client donné. À titre d'exemple, nous allons faire apparaître le chiffre d'affaires réalisé sur le mois de Mars avec le client Fabien. La procédure est la suivante :

À partir du TCD initialement obtenu, nous cliquons sur la flèche qui se trouve à côté du bouton « Clients » et sur la liste déroulante, nous décochons tous les clients à l'exception de Fabien.



Après validation de la sélection, nous obtenons le résultat ci-après présentant le chiffre d'affaires réalisé sur le mois de mars avec le client Fabien :

Filtrer		
Periode	Mars	
DATES	- tout -	
	Clients	
Données	Fabien	Total Résultat
Somme - R1.000	210 000	**210 000**
Somme - R2.000	474 000	**474 000**
Somme - R5.000	404 000	**404 000**
Somme - R10.000	878 000	**878 000**
Somme - R25.000	858 000	**858 000**

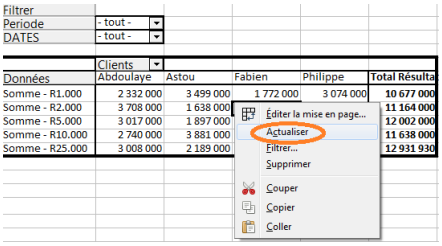
5 Modification des données sources et mise à jour du tableau croisé dynamique

Le tableau croisé dynamique finalement obtenu doit faire l'objet d'une mise à jour si une quelconque modification est faite au niveau des données sources. Cette mise à jour est réalisée en faisant un clic droit sur le tableau croisé dynamique puis, sur la fenêtre qui s'affiche, en cliquant sur « actualiser ».

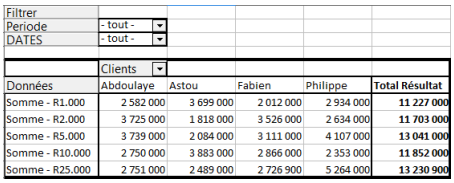
Exemple de modification : après création du TCD, nous nous sommes rendu compte que les données du mois de janvier figurant dans la table source étaient erronées et qu'il fallait prendre les éléments ci-après :

Periode	Clients	DATES	R1.000	R2.000	R5.000	R10.000	R25.000
Janvier	Abdoulaye	01/01/2013	500 000	600 000	900 000	900 000	640 000
Janvier	Astou	15/01/2013	400 000	830 000	470 000	950 000	420 000
Janvier	Fabien	27/01/2013	390 000	250 000	200 000	285 000	229 000
Janvier	Philippe	31/12/2013	200 000	500 000	1 000 000	333 000	700 000

Afin d'actualiser le TCD initialement créé, il faudra d'abord mettre à jour les données de la table source. Une fois le tableau source actualisé, nous faisons un clic droit sur le TCD et sur la fenêtre qui s'affiche, on choisit « Actualiser ».

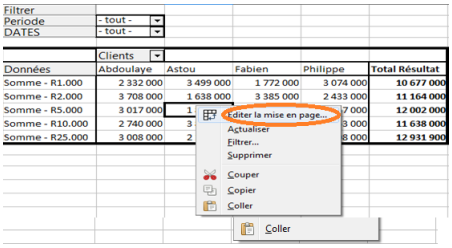


Après l'actualisation, nous obtenons le TCD mis à jour suivant :

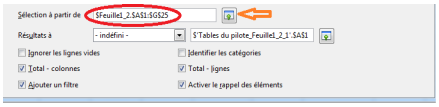


En cas de suppression ou d'ajout d'un nouveau champ, il faudra indiquer au tableur la nouvelle plage à prendre en compte afin de mettre à jour le TCD. Pour ce faire, il faudra procéder comme suit :

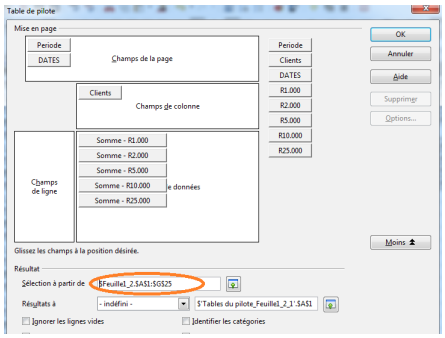
- après suppression d'un champ (dans notre cas, nous supprimons le champ R25.000 qui se trouve dans la colonne H du tableau source), nous positionnons la souris sur le TCD et effectuons un clic droit ;
- sur la fenêtre qui s'affiche, nous choisirons l'option « Éditer la mise en page » ;



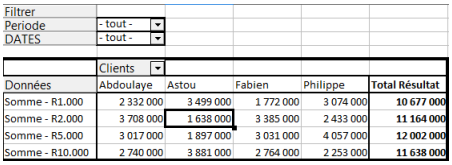
- sur l'assistant initial de création qui s'affiche, nous faisons un clic sur « Plus » et effaçons la présente sélection relative à l'ancienne plage. À la suite de cela, il nous faudra cliquer sur le bouton « Réduire » qui se trouve à droite de la sélection effacée ; cela nous permettra de définir la nouvelle plage des données sources :



- après définition de la nouvelle plage, il faudra cliquer sur le bouton « Entrer » de notre clavier et procéder à la fermeture de la fenêtre en cliquant sur le bouton « **Ok** ». La nouvelle zone est ainsi automatiquement prise en compte, comme le montre la figure ci-dessous :



- il faut ensuite se positionner sur le TCD initial et, comme décrit plus haut, faire un clic droit pour ensuite choisir « Actualiser ». Nous obtenons ainsi le TCD mis à jour ci-après :



Retrouvez l'article de ***Malick Seck*** *en ligne : lien 143*

LaTeX

Les derniers tutoriels et articles

Les index sous LaTeXet leur personnalisation

Cet article a pour but de faciliter la création d'index personnalisés à ceux qui le souhaitent.

1 Introduction

Utilisant depuis un moment déjà LaTeX, j'ai dû avoir recours à un index afin de faciliter les recherches dans un document qui prenait de l'ampleur. Quelle ne fut pas ma surprise de voir la tête des index par défaut qui ne sont guère attirants.

Je me suis donc plongé dans la documentation afin de voir ce que l'on pouvait faire pour obtenir des index plus attrayants.

Cet article devrait vous permettre de personnaliser vos index pas à pas. Je ne ferai pas une présentation de LaTeX, cet article présupposant que vous connaissez au minimum celui-ci et son fonctionnement.

2 Création d'un index

Dans LaTeX, générer un index est une chose très simple. Trois commandes sont principalement utilisées :

- \makeindex dans le préambule afin d'informer LaTeXqu'il faut générer l'index ;
- \index<mot ou expression à indexer> ;
- \printindex afin d'afficher l'index généré.

L'exemple ci-dessous montre l'utilisation de ces trois commandes ainsi que leur ordre d'apparition dans le document.

La commande \printindex peut indifféremment être placée en début ou en fin de document, même si on trouve plus généralement l'index en fin de document, LaTeXremplaçant \printindex par l'index formaté.

```
1  \documentclass[a4paper,12pt,francais]{
       article}
2  \usepackage[utf8]{inputenc}
3  \usepackage[T1]{fontenc}
4  \usepackage{lipsum}
5  \usepackage[francais]{babel}
6  \makeindex
7  \begin{document}
8  \section{Lipsum 1}\index{Lipsum 1}
9  \lipsum[1]
10 \section{Lipsum 2}\index{Lipsum 2}
11 \lipsum[2]
12 \printindex
13 \end{document}
```

2.1 Utilisation de la commande index

2.1.a Les caractères accentués

Les caractères accentués ne sont pas gérés par la commande makeindex.

Aussi, pour correctement classer l'entrée dans l'index, il convient de définir ce dernier de la manière suivante : \indexsans_accents@avec_accents.

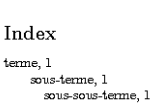
Dans l'exemple qui suit, le mot « accentué » sera classé comme « accentue ».

```
1  \index{accentue@accentué}
```

2.1.b Entrées et sous-entrées

Pour mémoire, la commande makeindex ne gère que trois niveaux d'entrées. Les différents niveaux d'entrées sont séparés par le caractère « **!** » (point d'exclamation).

1. **Entrée simple** : \indexterme.
2. **Entrée à deux niveaux** : \indexterme !sous-terme.
3. **Entrée à trois niveaux** : \indexterme !sous-terme !sous-sous-terme.


Index
terme, 1
sous-terme, 1
sous-sous-terme, 1

2.1.c Renvoi à une autre entrée

Il est possible de faire référence à une autre entrée de l'index pour un terme indexé. Cela se fait de la manière suivante : \indexThermes|seeBains.

Le caractère « | » est obtenu par la combinaison de touches « Alt Gr + 6 » sur un clavier azerty français.

\indexThermes|seeBains donnera dans l'index « **Thermes, voir Bains** ».

2.2 Génération d'un premier index

L'exemple ci-dessous montre l'utilisation des trois commandes évoquées plus haut, ainsi que leur ordre d'apparition dans le document LaTeX.

```
1  \documentclass[a4paper,12pt,francais]{
       article}
2  \usepackage[utf8]{inputenc}
3  \usepackage[T1]{fontenc}
4  \usepackage{lipsum}
5  \usepackage[francais]{babel}
6  \usepackage{makeidx} % création d'index
7  \makeindex
8  \begin{document}
9  \lipsum[1]\index{Lipsum 1}
10 \lipsum[2]\index{Lipsum 2}
11 \printindex
12 \end{document}
```

Pour lancer la compilation du fichier exemple.tex, on utilise les commandes suivantes :

```
1 latex (ou pdflatex) exemple.tex
2 makeindex exemple.idx
3 latex (ou pdflatex) exemple.tex
```

L'usage de la commande \makeindex dans le préambule du document LaTeXgénérera deux fichiers nommés dans notre cas **exemple.ilg** et **exemple.ind**, le fichier **exemple.idx** étant généré par la commande LaTeX(ou pdflatex). Le fichier exemple.ilg contient les logs de compilation, exemple.ind l'index formaté qui sera inclus dans le document final par la commande \printindex, le fichier exemple.idx contenant les entrées de l'index.

Rien de très sexy, il faut bien l'avouer.

Pour y remédier, nous allons personnaliser notre index à l'aide d'un fichier de gabarit que nous nommerons perso.ist et qui sera sauvegardé dans le même répertoire que notre fichier source LaTeX.

3 Personnalisation du fichier « perso.ist »

Actuellement, notre fichier perso.ist est vide.

Dans cette partie, nous allons petit à petit le remplir afin d'obtenir un index beaucoup plus agréable.

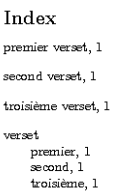
3.1 Le fichier exemple.tex

Nous allons commencer par apporter quelques modifications à notre fichier exemple.tex afin d'avoir plus de termes à indexer et dans une plus grande gamme.

Voici le fichier exemple.tex qui nous servira pour la suite :

```
1  \documentclass[a4paper,10pt,francais]{
       article}
2  \usepackage[utf8]{inputenc}
3  \usepackage[T1]{fontenc}
4  \usepackage[francais]{babel}
5  \usepackage{color}  % définitions des
       couleurs
6  \usepackage[dvipsnames,x11names,svgnames
       ,table]{xcolor} % texte et tableau
       en couleur
7  \usepackage{makeidx} % création d'index
8  \usepackage{lipsum}
9  \makeindex
10 \begin{document}
11 \lipsum[1]\index{premier verset}\index{
       verset!premier}
12 \lipsum[2]\index{second verset}\index{
       verset!second}
13 \lipsum[3]\index{troisieme verset@troisi
       ème verset}\index{verset!
       troisieme@troisième}
14 \printindex
15 \end{document}
```

Résultat de compilation



Comme vous pouvez le voir, les termes sont indexés deux fois : une première fois avec une entrée simple et une seconde fois avec une sous-entrée. Ceci est bien sûr volontaire et nous permettra de mettre en place et de comprendre la mise en forme des sous-entrées dans le gabarit de l'index.

3.2 Faire apparaître les en-têtes de groupe de l'index

Pour cela, nous allons ajouter une ligne à notre fichier perso.ist :

```
1 headings_flag 1
```

La valeur de headings_flag peut-être -1, 0 ou 1 selon le résultat voulu.

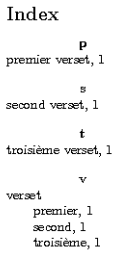
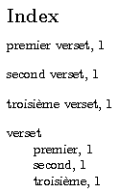
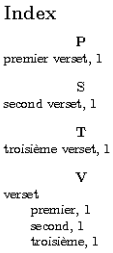
— headings_flag : selon la valeur choisie (-1, 0 ou 1) les en-têtes de groupe seront respectivement en minuscules, aucun en-tête ou en majuscules.

Le fait d'utiliser un fichier ist personnalisé modifie notre chaîne de compilation de la manière suivante :

```
1 latex (ou pdflatex) exemple.tex
2 makeindex exemple.idx -s perso.ist
3 latex (ou pdflatex) exemple.tex
```

L'ajout de **-s perso.ist** à la fin de la commande **makeindex** indique où se situe notre fichier personnalisé ainsi que son nom.

Valeurs et résultats de headings_flag :

-1 : En-têtes en minuscules	0 : Aucun en-tête	1 : En-têtes en majuscules
		

3.3 Améliorer les en-têtes de groupe de l'index

Les en-têtes de groupe que nous avons ne sont pas particulièrement attirants. Nous allons améliorer leur apparence ici.

Pour cela, nous allons insérer les lignes ci-dessous dans notre fichier perso.ist :

```
1 heading_prefix "{\\bfseries\\hfil "
2 heading_suffix "\\hfil}\\nopagebreak\n"
```

Une petite explication s'impose.

Les deux lignes ci-dessus forment une suite de commandes LaTeXqui seront exécutées avant (pour heading_prefix) et après (pour heading_suffix) les en-têtes de groupe.

— heading_prefix : commande ou suite de commandes exécutées avant les en-têtes de groupe.

— heading_suffix : commande ou suite de commandes exécutées après les en-têtes de groupe.

Il faut bien comprendre que ces deux lignes de commandes s'agrègent pour n'en former qu'une lors de la mise en forme de l'index.

Il est aussi important de noter que :

— une chaîne de caractères doit être mise entre guillemets ("<chaine>") ;

— la longueur maximale d'une chaîne de caractères est de 2048 ;

— pour les commandes LaTeX, l'antislash (\) doit être doublé ;

— la chaîne '\n' permet d'insérer un retour ligne ;

— la chaîne '\t' permet d'insérer une tabulation.

Ainsi, lors de la mise en forme de l'index, elles donnent la ligne suivante :

```
1 {\bfseries\hfil <Lettre d'en-tête de
    groupe> \hfil}\nopagebreak
```

3.3.a Explications

Les diverses commandes utilisées ci-dessus sont normalement connues d'un utilisateur de LaTeX :

— un groupe est créé avec ... afin de limiter la portée des commandes au seul en-tête de groupe ;

— \bfseries permet d'activer la mise en gras ;

— \hfil insère un espace élastique ;

— À cet endroit, sera insérée la lettre correspondant à l'en-tête de groupe ;

— \hfil insère un espace élastique ;

— \nopagebreak indique que l'on ne souhaite pas de saut de page à cet endroit.

```
1 {\bfseries\hfil t \hfil}\nopagebreak
```

```
1 {\bfseries\hfil T \hfil}\nopagebreak
```

Il est possible d'ouvrir un groupe dans heading_prefix et de le fermer dans heading_suffix. Toutefois, attention à bien le fermer sinon des erreurs de compilation apparaîtront.

La façon la plus simple est, à mon avis, de concevoir la ligne de commande entière, puis de la couper une fois qu'elle est fonctionnelle et enfin de l'insérer dans le fichier **ist** personnalisé.

3.3.b Exemples divers

```
1 heading_prefix "{\\bfseries\\hfil \\fbox
    { "
2 heading_suffix "}\\hfil}\\nopagebreak\n"
```

```
1 heading_prefix "{\\bfseries\\hfil\\
    textcolor{blue}{ "
2 heading_suffix "}\\hfil}\\nopagebreak\n"
```

```
1 heading_prefix "{\\bfseries\\hfil\\fbox{
    \\textcolor{blue}{ "
2 heading_suffix "}}\\hfil}\\nopagebreak\n
    "
```

```
1 heading_prefix "{\\bfseries\\hfil\\
    fcolorbox{red}{lightgray}{\\
    textcolor{blue}{ "
2 heading_suffix "}}\\hfil}\\nopagebreak\n
    "
```

3.4 Et pour les nombres et les symboles ?

Pour les nombres et les symboles, les commandes sont les suivantes :

— symhead_positive <string> : affiche la chaîne <string> si headings_flag est positif ;

— symhead_negative <string> : affiche la chaîne <string> si headings_flag est négatif ;

— numhead_positive <string> : affiche la chaîne <string> si headings_flag est positif ;

— numhead_negative <string> : affiche la chaîne <string> si headings_flag est négatif.

En résumé :

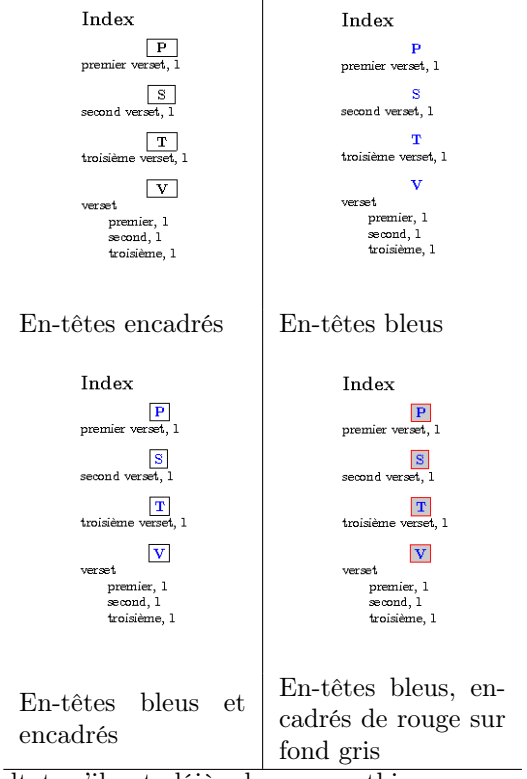
— si headings_flag est négatif (en-têtes de groupe en minuscules), les chaînes de symhead_negative et numhead_negative sont utilisées, respectivement pour les symboles et les nombres.

— si headings_flag est positif (en-têtes de groupe en majuscules) les chaînes de symhead_positive et numhead_positive sont utilisées, respectivement pour les symboles et les nombres.

Comme par défaut le texte est en anglais, nous allons le personnaliser en français en rajoutant les lignes suivantes à notre fichier perso.ist :

```
1 symhead_positive "Symboles"
2 symhead_negative "symboles"
3 numhead_positive "Nombres"
4 numhead_negative "nombres"
```

Résultat des compilations :



Le résultat, s'il est déjà plus sympathique, a encore besoin d'améliorations. En effet, notre en-tête de groupe est pratiquement collé à la ligne qui le suit.

Pour corriger ce petit défaut, il suffit de modifier légèrement la ligne heading_suffix de notre fichier perso.ist en ajoutant l'instruction \\vspace*3ex entre \\hfil et \\nopagebreak\n, ce qui aérera notre index.

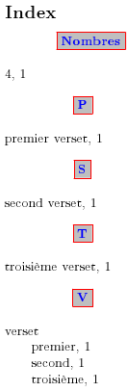
```
1 heading_suffix "\\hfil\\vspace*{3ex}\\
    nopagebreak\n"
```

Nous rajoutons aussi la ligne suivante à notre fichier exemple.tex :

```
1 \lipsum[4]\index{4}
```

Une fois compilé, on obtient le résultat ci-dessous. C'est déjà beaucoup plus agréable non ?

Résultat de la compilation



3.5 Améliorer la présentation des termes indexés et des numéros de pages

3.5.a Les points de suite

Personnellement, j'aime bien qu'entre le terme indexé et les numéros de pages, il y ait des points de suite. Je trouve que cela facilite la lecture.

Je vais donc vous montrer comment obtenir cela.

La liste qui suit vous permettra de comprendre chaque paramètre :

— delim_0 <string> : chaîne qui sera insérée entre le terme indexé de niveau 0 et le premier numéro de page, par défaut une virgule suivie d'une espace (", ") ;

— delim_1 <string> : chaîne qui sera insérée entre le terme indexé de niveau 1 et le premier numéro de page, par défaut une virgule suivie d'une espace (", ") ;

— delim_2 <string> : chaîne qui sera insérée entre le terme indexé de niveau 2 et le premier numéro de page, par défaut une virgule suivie d'une espace (", ") ;

— delim_n <string> : chaîne qui sera insérée entre deux numéros de page pour le même terme indexé, par défaut une virgule suivie d'une espace (", ") ;

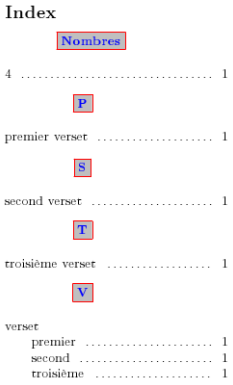
— delim_r <string> : chaîne qui sera insérée entre le premier et le dernier numéro de page d'une plage, par défaut "–" ;

— delim_t <string> : chaîne qui sera insérée à la fin d'une liste de numéro de page, par défaut "". Cette chaîne n'aura aucun effet s'il n'y a aucune liste de numéros de page.

Fort de ces informations, il ne reste plus qu'à insérer les trois lignes ci-dessous dans notre fichier perso.ist et lancer une compilation :

```
1 delim_0 "\\hspace{6pt}\\dotfill\\hspace{
    6pt}"
2 delim_1 "\\hspace{6pt}\\dotfill\\hspace{
    6pt}"
3 delim_2 "\\hspace{6pt}\\dotfill\\hspace{
    6pt}"
```

Résultat de la compilation



Joli résultat. À un détail près. Notre index semble ne prendre que la moitié de la largeur de la page. Rien de bien grave, le style de page utilisé pour l'index est le style **plain** en mode deux colonnes (personnellement, j'aime bien en deux colonnes). Peut-être qu'avec le package **multicol**, il est possible de passer en mode une ou trois colonnes. Je n'ai pas testé. Si quelqu'un sait comment faire, je suis preneur de l'information.

Si vous souhaitez redéfinir l'en-tête ou le pied de page, le package **fancyhdr** vous permettra de le faire sans difficultés.

3.5.b Les termes indexés

La liste ci-dessous fournit toutes les informations permettant de mettre en forme les termes indexés :

- item_0 <string> : commande insérée entre deux niveaux primaires de l'index (level 0). Par défaut '\n \\item ' ;
- item_1 <string> : commande insérée entre deux niveaux secondaires de l'index (level 1). Par défaut '\n \\subitem ' ;
- item_2 <string> : commande insérée entre deux niveaux tertiaires de l'index (level 2). Par défaut '\n \\subsubitem ' ;
- item_01 <string> : commande insérée entre un niveau primaire (level 0) et un niveau secondaire (level 1) de l'index. Par défaut '\n \\subitem ' ;
- item_x1 <string> : commande insérée entre un niveau primaire (level 0) et un niveau secondaire (level 1) de l'index quand le niveau primaire n'a pas de numéro de page associé. Par défaut '\n \\subitem ' ;
- item_12 <string> : commande insérée entre un niveau secondaire (level 1) et un niveau tertiaire (level 2) de l'index. Par défaut '\n \\subsubitem ' ;
- item_x2 <string> : commande insérée entre un niveau secondaire (level 1) et un niveau tertiaire (level 2) de l'index quand le niveau secondaire n'a pas de numéro de page associé. Par défaut '\n \\subsubitem '.

Afin de faciliter la compréhension, nous allons ajouter les deux lignes suivantes dans notre fichier perso.ist :

```
1 item_0 "\n \\item \\bfseries{"
2 item_x1 "}\\normalfont\n \\subitem "
```

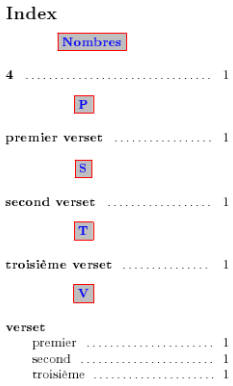
Nous modifions aussi la ligne suivante :

```
1 delim_0 "}\\hspace{6pt}\\dotfill\\hspace
    {6pt}"
```

Avec ces modifications, les termes indexés de premier niveau seront en gras, tandis que les termes des deuxième et troisième niveaux seront en graisse normale.

Voici le résultat de la compilation prenant en compte les modifications précédentes.

Résultat de la compilation



Ci-dessous le fichier exemple.ist final ayant permis la génération de l'index affiché au-dessus :

```
1  heading_prefix "{\\bfseries\\hfil\\
     fcolorbox{red}{lightgray}{\\
     textcolor{blue}{ "
2  heading_suffix "}}\\hfil}\\vspace*{3ex}
     \\nopagebreak\n"
3  headings_flag 1
4  symhead_positive "Symboles"
5  symhead_negative "symboles"
6  numhead_positive "Nombres"
7  numhead_negative "nombres"
8  delim_0 "}\\hspace{6pt}\\dotfill\\hspace
     {6pt}"
9  delim_1 "\\hspace{5pt}\\dotfill\\hspace{
     5pt}"
10 delim_2 "\\hspace{4pt}\\dotfill\\hspace{
     4pt}"
11 item_0 "\n \\item {\\bfseries "
12 item_x1 "}\n \\normalfont\\subitem "
```

4 Un index c'est bien, plusieurs c'est mieux

Il peut parfois être pratique d'avoir une indexation par thème. Le package **index** permet de gérer des index multiples, ce qui autorise une indexation par thème très facilement.

Voici la marche à suivre :

1. L'appel du package **index** doit se faire dans le préambule après l'appel du package **makeidx** ;
2. La définition du nouvel index doit se faire avant la commande **makeindex**.

Voici la définition d'un nouvel index qui regroupera la liste des auteurs apparaissant dans un document :

```
\usepackage{makeidx}
\usepackage{index}
\newindex{aut}{otx}{otd}{Liste des
    auteurs}
```

Comme le montre l'extrait de notre fichier exemple.tex ci-dessus, le package **index** est appelé après le package **makeidx**, puis suit la définition d'un nouvel index.

Cet ordre est indispensable, le package **index** redéfinissant la commande index du package **makeidx** ainsi que la commande \printindex.

La commande \newindex est composée de quatre champs :

— aut : identifiant de l'index ;
— otx : extension du fichier contenant les entrées de l'index **aut** (exemple.otx) ;
— otd : extension du fichier contenant l'index **aut** formaté (exemple.otd) ;
— Liste des auteurs : libellé apparaissant comme titre de l'index personnalisé.

```
\documentclass[a4paper,10pt,francais]{
    article}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[francais]{babel}
\usepackage{color}  % d'é'finitions des
    couleurs
\usepackage[dvipsnames,x11names,svgnames
    ,table]{xcolor} % texte et tableau
    en couleur
\usepackage{makeidx} % cr'é'ation d'
    index
\usepackage{index} % cr'é'ation d'index
    multiples
\newindex{aut}{otx}{otd}{Liste des
    auteurs} % définition d'un nouvel
    index personnalisé
\usepackage{lipsum}
\makeindex
\begin{document}
\lipsum[1]\index{premier verset}\index{
    verset!premier}
\lipsum[2]\index{second verset}\index{
    verset!second}
\lipsum[3]\index{troisieme verset@troisi
    ème verset}\index{verset!
    troisieme@troisième}
\lipsum[4]\index{4}\index[aut]{moimoi}
\printindex
\printindex[aut]
\end{document}
```

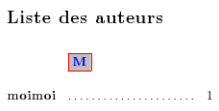
Pour pouvoir compiler notre index personnalisé ainsi que notre index habituel, voici la chaîne de compilation nécessaire :

```
latex (ou pdflatex) exemple.tex
makeindex -s perso.ist exemple.idx
makeindex.exe exemple.otx -t exemple.otg
     -s perso.ist -o exemple.otd
latex (ou pdflatex) exemple.tex
```

Le fichier exemple.otg (non défini dans la déclaration du nouvel index) n'est autre que le fichier de log de la création de l'index (très utile à lire si la génération de l'index ne fonctionne pas bien).

Voici le résultat de la compilation prenant en compte les modifications précédentes.

Résultat de la compilation



5 Conclusion

Comme vous avez pu le voir au cours de cet article, personnaliser l'apparence de son index n'est pas particulièrement difficile une fois que l'on a compris comment faire. Toutefois, il est prudent d'avancer pas à pas et d'effectuer des compilations régulières afin de juger du résultat obtenu et si celui-ci correspond bien à nos attentes.

Retrouvez l'article de ***Christophe Boulet*** *en ligne : lien 144*

Liste des liens

Page 2
lien 1 : ... http://asm.developpez.com/telecharger/detail/id/4101/Assembleur-en-ligne-avec-le-langage-C-et-le-compilateur-GCC

Page 3
lien 2 : ... https://www.kernel.org/pub/linux/kernel/Historic
lien 3 : ... http://asm.developpez.com/cours/gas/

Page 5
lien 4 : ... https://gcc.gnu.org/onlinedocs/gcc-4.9.0/gcc/Constraints.html#Constraints

Page 10
lien 5 : ... http://asm.developpez.com/cours/asminline/

Page 12
lien 6 : ... http://www.eclipse.org/downloads/
lien 7 : ... https://www.youtube.com/watch?feature=player_embedded&v=uk5XFUj6vxY
lien 8 : ... http://www.developpez.net/forums/d1451852/environnements-developpement/eclipse/nouvelle-version-l-environnement-eclipse-disponible-eclipse-luna-apporte-support-natif-java-8-a/
lien 9 : ... http://www.developpez.com/actu/68803/Eclipse-Orion-5-sort-avec-un-nouveau-look-et-plusieurs-changements-l-EDI-Web-dans-le-Cloud-permet-de-deployer-directement-sur-les-plateformes-Cloud/
lien 10 : ... http://planetorion.org/news/2014/06/orion-6-0-language-tooling-enhancements/
lien 11 : ... http://www.developpez.net/forums/d1454205/environnements-developpement/eclipse/eclipse-orion-6-debarque-support-ameliore-javascript/

Page 13
lien 12 : ... http://alain-bernard.developpez.com/tutoriels/eclipse/sirius-intro/fichiers/src-intro-sirius.zip
lien 13 : ... http://alain-bernard.developpez.com/tutoriels/eclipse/sirius-intro/fichiers/src-intro-sirius.zip
lien 14 : ... http://mbaron.developpez.com/tutoriels/eclipse/emf/creation-instanciation-modeles/
lien 15 : ... http://wiki.eclipse.org/Sirius/Update_Sites

Page 17
lien 16 : ... http://www.eclipse.org/acceleo/

Page 21
lien 17 : ... http://www.eclipse.org/sirius/
lien 18 : ... http://www.eclipse.org/sirius/doc/specifier/Sirius%20Specifier%20Manual.html
lien 19 : ... http://www.eclipse.org/acceleo/
lien 20 : ... http://wiki.eclipse.org/EEF
lien 21 : ... http://alain-bernard.developpez.com/tutoriels/eclipse/sirius-intro/

Page 22
lien 22 : ... http://mreinhold.org/blog/jigsaw-phase-two
lien 23 : ... http://openjdk.java.net/projects/jigsaw/goals-reqs/03
lien 24 : ... http://www.developpez.net/forums/d1455040/java/general-java/java-projet-jigsaw-se-concretise/

Page 23
lien 25 : ... http://www.easybatch.org/

Page 25
lien 26 : ... https://gist.github.com/benas/8547412
lien 27 : ... ./fichiers/easybatch-dvp.zip

Page 26
lien 28 : ... http://mahmoudbenhassine.wordpress.com/2014/03/03/spring-batch-vs-easy-batch-a-hello-world-comparison/

Page 27

lien 29 : ... https://github.com/benas/easy-batch
lien 30 : ... http://www.easybatch.org
lien 31 : ... https://speakerdeck.com/benas/easy-batch
lien 32 : ... http://benassi.developpez.com/tutoriels/java/developper-batch-easybatch-5-minutes/

Page 37

lien 33 : ... https://www.meteor.com/
lien 34 : ... https://www.discovermeteor.com/
lien 35 : ... https://www.eventedmind.com/
lien 36 : ... http://meteorhacks.com/
lien 37 : ... http://www.meteorpedia.com/
lien 38 : ... http://soren-ohnmeiss.developpez.com/tutoriels/javascript/meteor/introduction-meteor/

Page 38

lien 39 : ... http://web.developpez.com/livres/index/?page=Conception-generale-de-sites-web#L2212139861
lien 40 : ... http://adage.com/article/digital/facebook-41-ad-revenue-mobile-q2/243293/
lien 41 : ... http://slideshare.net/MicrosoftTag
lien 42 : ... http://gs.statcounter.com/#mobile_vs_desktop-IN-monthly-201101-201309

Page 42

lien 43 : ... http://web.developpez.com/livres/index/?page=Conception-generale-de-sites-web#L2212139861
lien 44 : ... http://www.editions-eyrolles.com/Livre/9782212139860/creer-un-seul-site-pour-toutes-les-plates-formes
lien 45 : ... http://web.developpez.com/livres/index/?page=Conception-generale-de-sites-web#L2212139861
lien 46 : ... http://sylvainpv.developpez.com/livres/web/creer-un-seul-site-pour-toutes-les-plates-formes/

Page 43

lien 47 : ... http://imikado.developpez.com/tutoriels/php/presentation-mkframework/
lien 48 : ... http://php.net/manual/fr/function.sha1.php

Page 48

lien 49 : ... http://mkframework.com/

Page 49

lien 50 : ... http://imikado.developpez.com/tutoriels/php/creer-gestionnaire-contact-googleMaps/

Page 50

lien 51 : ... https://github.com/google/traceur-compiler
lien 52 : ... https://www.dartlang.org/

Page 51

lien 53 : ... https://github.com/lukehoban/es6features
lien 54 : ... http://coffeescript.org/
lien 55 : ... http://blogs.msdn.com/b/officeapps/archive/2013/03/18/excel-does-javascript-a-vba-developer-s-perspective.aspx
lien 56 : ... http://www.typescriptlang.org/Playground/

Page 52

lien 57 : ... http://www.typescriptlang.org/Handbook
lien 58 : ... http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf
lien 59 : ... http://www.developpez.com/actu/69143/Facebook-presente-Hack-son-langage-de-programmation-derive-de-PHP-qui-apporte-plusieurs-nouveautes/
lien 60 : ... http://definitelytyped.org/
lien 61 : ... http://nodejs.org/
lien 62 : ... http://www.typescriptlang.org/
lien 63 : ... http://www.jetbrains.com/webide/
lien 64 : ... https://github.com/palantir/eclipse-typescript

Page 53

lien 65 : ... http://www.typescriptlang.org/Handbook
lien 66 : ... http://www.developpez.net/forums/f2001/webmasters-developpement-web/javascript-ajax-typescript-dart/typescript/
lien 67 : ... http://tarh.developpez.com/articles/typescript/pourquoi-utiliser-typescript/

Page 54
lien 68 : ... https://www.scirra.com/demos/ghosttutorial/
lien 69 : ... https://www.scirra.com/tutorials/253/how-to-make-a-platform-game
lien 70 : ... https://www.scirra.com/tutorials/358/asteroid-clone-in-less-than-100-events
lien 71 : ... https://www.scirra.com/construct2/releases/new

Page 65
lien 72 : ... https://www.scirra.com/tutorials/42/upload-your-game-to-dropbox
lien 73 : ... https://www.scirra.com/downloads/ghostshooter-tutorial.capx
lien 74 : ... https://www.scirra.com/manual/84/sounds-music
lien 75 : ... https://www.scirra.com/tutorials/253/how-to-make-a-platform-game
lien 76 : ... https://www.scirra.com/manual/75/how-events-work
lien 77 : ... https://www.scirra.com/manual/1/construct-2

Page 66
lien 78 : ... http://jeux.developpez.com/tutoriels/construct2/guide-debutant/

Page 70
lien 79 : ... http://jeux.developpez.com/tutoriels/graphisme-2d-programmeurs/commencons-cercles/

Page 71
lien 80 : ... http://gamelier.org/
lien 81 : ... http://cri-paris.org/
lien 82 : ... http://www.metacritic.com/game/wii-u/rayman-legends

Page 74
lien 83 : ... http://jeux.developpez.com/making-of/rayman-legends/musique-design/

Page 75
lien 84 : ... http://www.docbook.org/
lien 85 : ... http://www.debian.org/
lien 86 : ... http://www.inetdoc.net/pdf/rnis.pdf
lien 87 : ... http://www.itu.int/

Page 76
lien 88 : ... http://www.etsi.org/
lien 89 : ... http://www.inetdoc.net/articles/rnis/rnis.how.html#rnis.how.intf.bri
lien 90 : ... http://www.inetdoc.net/articles/rnis/rnis.how.html#rnis.how.intf.pri

Page 77
lien 91 : ... http://www.inetdoc.net/articles/rnis/rnis.how.html#rnis.how.intf.bri
lien 92 : ... http://www.inetdoc.net/articles/rnis/rnis.how.html#rnis.how.intf.pri
lien 93 : ... http://www.tldp.org/HOWTO/PPP-HOWTO/
lien 94 : ... http://www.faqs.org/rfcs/rfc1717.html
lien 95 : ... http://www.faqs.org/rfcs/rfc1990.html

Page 78
lien 96 : ... http://www.inetdoc.net/articles/rnis/images/protocoles.png
lien 97 : ... http://www.inetdoc.net/articles/rnis/images/frames.png

Page 79
lien 98 : ... http://www.inetdoc.net/articles/rnis/images/hdlc.png

Page 81
lien 99 : ... http://www.inetdoc.net/articles/rnis/images/D-call.png
lien 100 : ... http://fbrunel.free.fr/isdnfaq.pdf
lien 101 : ... http://inetdoc.developpez.com/tutoriels/technologie-rnis/

Page 82
lien 102 : ... http://www.docbook.org/
lien 103 : ... http://www.debian.org/

lien 104 : ... http://www.inetdoc.net/pdf/modelisations.pdf

Page 86
lien 105 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.sna
lien 106 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.iso
lien 107 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.osi
lien 108 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.arpanet
lien 109 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.itu
lien 110 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.arcep
lien 111 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.ieee
lien 112 : ... http://www.openldap.org/

Page 87
lien 113 : ... http://www.inetdoc.net/articles/modelisation/modelisations.concept.html#error-control
lien 114 : ... http://www.inetdoc.net/articles/modelisation/modelisations.concept.html#flow-control

Page 88
lien 115 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.arpanet
lien 116 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.osi
lien 117 : ... http://fr.wikipedia.org/wiki/Request_for_comments

Page 89
lien 118 : ... http://www.inetdoc.net/articles/modelisation/modelisations.contemporain.html#rfc.protocol.stack
lien 119 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.arpanet
lien 120 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.osi
lien 121 : ... http://www.faqs.org/rfcs/rfc791.html
lien 122 : ... http://www.inetdoc.net/guides/lartc/

Page 90
lien 123 : ... http://www.inetdoc.net/articles/adressage.ipv4/
lien 124 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.arpanet
lien 125 : ... http://www.faqs.org/rfcs/rfc793.html

Page 92
lien 126 : ... http://www.inetdoc.net/articles/modelisation/modelisations.conclusion.html#glossaire.ieee
lien 127 : ... http://www.faqs.org/rfcs/rfc894.html
lien 128 : ... http://www.faqs.org/rfcs/rfc1042.html
lien 129 : ... http://www.iana.org/assignments/ethernet-numbers/ethernet-numbers.xhtml
lien 130 : ... http://www.inetdoc.net/articles/modelisation/modelisations.osi.html
lien 131 : ... http://www.inetdoc.net/articles/modelisation/modelisations.tcpip.html
lien 132 : ... http://www.faqs.org/rfcs/rfc793.html
lien 133 : ... http://www.faqs.org/rfcs/rfc791.html

Page 93
lien 134 : ... http://www.inetdoc.net/articles/ethernet/
lien 135 : ... http://www.inetdoc.net/articles/adressage.ipv4/
lien 136 : ... http://www.inetdoc.net/travaux_pratiques/intro.analyse/
lien 137 : ... http://www.arcep.fr/
lien 138 : ... http://www.ieee.org/index.html
lien 139 : ... http://www.iso.org/iso/home.html
lien 140 : ... http://www.itu.int/en/Pages/default.aspx
lien 141 : ... http://standards.iso.org/ittf/licence.html
lien 142 : ... http://inetdoc.developpez.com/tutoriels/modelisation-reseau/

Page 98
lien 143 : ... http://malick-nseck.developpez.com/tutoriels/apprendre-a-creer-tableau-croise-dynamique-avec-libre-office-calc/

Page 104
lien 144 : ... http://winnt.developpez.com/tutoriels/latex-index/