Developpez

Le Mag

Edition de Avril - Mai 2010.
Numéro 27.
Magazine en ligne gratuit.

Réalisation : Alexandre Pottiez
Rédaction : la rédaction de Developpez
Contact : magazine@redaction-developpez.com

Sommaire

Editorial

De nouvelles technologies sont sorties pendant ces deux derniers mois, il n'en fallait pas plus à nos rédacteurs pour qu'ils rédigent à votre intention le meilleur du meilleur : une toute nouvelle FAQ Android, une large partie dédiée aux méthodes Agiles, etc. Et qui a dit que Quake ne pouvait pas être un support de test de nouvelles technos ? La preuve en code...

La rédaction

Article Conception

Compte-rendu XP Day Suisse 2010

Retrouvez le meilleur des XP Day Suisse 2010.

par **Bruno Orsier**
Page 47

Article (X)HTML/CSS

Quake 2 en JavaScript grâce à HTML5

Retrouvez votre jeu préféré directement sur votre navigateur Web.

par **David Delbecq**
Page 15

Raccourcis clavier avec SWT StyledText

Dans ce billet nous allons présenter une solution pour incorporer les actions Copy/Paste et Undo/Redo au widget StyledText (Lien1). Ce widget fait partie de la librairie SWT (Lien2), qui fournit un toolkit de composants graphiques pour développer une interface graphique en Java.
De plus, ce widget s'intègre dans des éditeurs de l'environnement Eclipse RCP (Lien3), il est donc nécessaire que ces actions soient liées à cet environnement.

1. Présentation

Nous développons une application Eclipse RCP dans laquelle nous utilisons le pattern Master/Details page, avec les UI Forms (Lien4). Cela nous permet d'afficher nos données sous forme d'arbres et de pages détaillés, contenant les propriétés de l'élément édité. Dans les Details Page, nous utilisons donc des StyledText, pour renseigner les valeurs des différentes propriétés. Le choix du StyledText pour les champs d'édition s'explique par le fait qu'il s'agit d'un widget customisable qui permet de modifier la fonte ainsi que la couleur du texte affiché.

Figure 1 : Exemple de StyledText

2. La problématique

La problématique est donc de mettre en place les actions Copy/Paste et Undo/Redo dans nos différents StyleText, et que, de plus, celles-ci soient incorporées à notre RCP, via le menu *Edit*, comme n'importe quelle autre action disponible, comme par exemple dans les arbres.

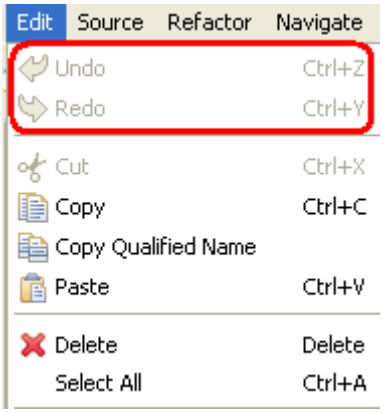


Figure 2 : Actions dans le menu Edit

Cependant, le StyledText ne dispose pas de ces actions, que l'on peut avoir habituellement dans les champs texte, que ce soit par le biais d'un menu contextuel ou de raccourcis clavier, ce qui est donc problématique pour renseigner les différents champs que l'on retrouve dans notre application. Ceci s'explique par le fait que le StyledText est non natif, il ne bénéficie donc pas des actions par défaut, que l'on peut retrouver sur les widgets de gestion de texte. Il s'agit d'un "widget émulé, pour être utilisé dans les éditeurs Eclipse".

Nous souhaitons donc pouvoir disposer de ces actions dans nos widgets, que ce soit directement avec les raccourcis clavier, ou avec un menu contextuel sur le StyledText.

3. La solution mise en place

Pour résoudre ce problème, la solution mise en place est la suivante :

tout d'abord, nous créons la classe *MyStyledText*, dans laquelle nous avons un StyledText. Cette classe nous permet de gérer les listeners, ainsi que les actions que nous voulons ajouter au StyledText.

Nous avons trois types de listeners :

- ModifyListener : qui prend en compte la modification au niveau de l'élément, et informe l'éditeur qu'il a été modifié ;
- FocusListener : qui gère les actions Copy/Paste et Undo/Redo en fonction du widget qui prend le focus.
  Lorsqu'un StyledText prend le focus, nous modifions les actions de l'ActionBars pour que celles-ci lui soient applicables. Ensuite, lors de la perte du focus du StyledText, nous restaurons les actions de l'ActionBarContributor. En réaffectant les actions de l'ActionBars avec les actions du StyledText, nos actions sont donc intégrées à l'application RCP et sont de ce fait, accessibles dans la barre de menu, via la section *Edit*.

```
myStyledText.addFocusListener(new FocusListener()
{

        IAction undoActionOld;
        IAction redoActionOld;
        ...

        public void focusLost(FocusEvent e) {

                // Restore previous commands
                IActionBars actionBars =
getEditor().getEditorSite().getActionBars();
                actionBars.setGlobalActionHandler
(ActionFactory.UNDO.getId(),  undoActionOld);
                actionBars.setGlobalActionHandler
(ActionFactory.REDO.getId(),  redoActionOld);
                ...
                actionBars.updateActionBars();
```

```
        }

        public void focusGained(FocusEvent e) {

                // Stock previous commands
                undoActionOld =
getActionBars().getGlobalActionHandler(ActionFact
ory.UNDO.getId());
                redoActionOld =
getActionBars().getGlobalActionHandler(ActionFact
ory.REDO.getId());


                ...

                // Create the undo action
                undoAction= new
UndoActionHandler(getEditor().getSite(),
undoManager.getUndoContext());
                actionBars.setGlobalActionHandler
(ActionFactory.UNDO.getId(), undoAction);
                // Create the redo action.
                redoAction= new
RedoActionHandler(getEditor().getSite(),
undoManager.getUndoContext());
                actionBars.setGlobalActionHandler
(ActionFactory.REDO.getId(), redoAction);

                actionBars.updateActionBars();
        }
});
```

Pour les actions d'Undo/Redo du StyledText, nous avons créé la classe *UndoStyledTextManager*, qui permet de gérer les actions effectuées dans le widget. Pour cela, elle contient un *IUndoContext* (qui filtre les opérations sur lesquelles le undo/redo est possible) et un *IOperationHistory* (les actions sont ajoutées à l'historique une fois qu'elles ont été exécutées) des opérations disponibles sur le widget.

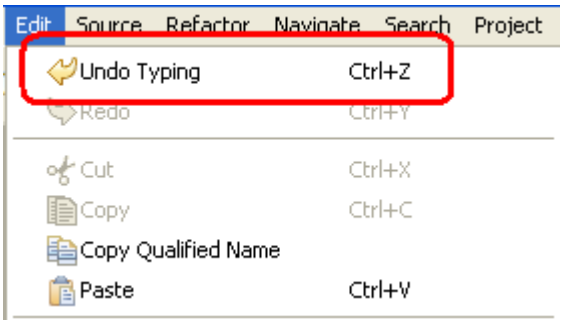


Figure 3 : Liaison avec le menu Edit

- MouseListener : qui gère le menu contextuel pour les actions sur le contenu du StyledText La classe *PopUpMenu* est couplée avec les actions construites précédemment.

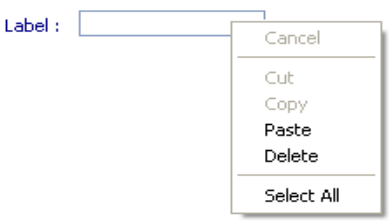


Figure 4 : Le menu contextuel

```
//Create the menu
menu = new Menu(parent.getShell(),SWT.POP_UP);
menuItemCancel = new MenuItem(menu, SWT.CASCADE);
menuItemSeparator = new MenuItem(menu,
SWT.SEPARATOR);
menuItemSeparator.setEnabled(true);
menuItemCut = new MenuItem(menu, SWT.CASCADE);
menuItemCopy = new MenuItem(menu, SWT.CASCADE);
menuItemPaste = new MenuItem(menu, SWT.CASCADE);
menuItemDelete = new MenuItem(menu, SWT.CASCADE);

...

menuItemCancel.addListener(SWT.Selection, new
Listener() {
  public void handleEvent(Event event) {
    setActionCancel(true);
    //Get the StyledText undo action
    myStyledText.getUndoAction.run();
    setMenuState(0);
  }
});

...
```

Ensuite, nous avons la classe *UndoStyledTextManager*, qui va gérer le contexte et l'historique des opérations lors des modifications du contenu du StyledText. Le widget sera connecté à ce manager durant son existence.

```
UndoStyledTextManager undoManager = new
UndoStyledTextManager(undoSize);
undoManager.connect(myStyledText);
```

Grâce à cette connexion, nous lui associons le contexte. Il s'agit de connaitre le contexte de l'action courante, pour déterminer les opérations à réaliser pour effectuer le Undo. De plus, nous avons également l'historique qui va gérer celui des opérations possibles suivant l'action courante. Celui-ci est géré par le biais d'un écouteur sur le StyledText. En effet, à chaque modification de son contenu, nous créons une nouvelle opération et nous l'ajoutons à l'historique. Il s'agit de nos propres opérations de type *UndoableOperation*. Celles-ci permettent d'enregistrer les modifications apportées au contenu du StyledText pour pouvoir ensuite bénéficier des actions de Undo et de Redo.

4. Evolution

L'axe d'évolution concerne la gestion des actions Undo/Redo, afin que celles-ci soient similaires à un éditeur de texte. En effet, actuellement ces actions agissent caractère par caractère, et non par suite de caractères tapés ou supprimés. Il pourrait donc être envisageable d'améliorer les actions de l'*UndoableOperation*.

Retrouvez l'article de David Chautard en ligne : Lien5

Android

La FAQ Android

La nouvelle FAQ Android vient de sortir, découvrez dès maintenant une sélection de questions-réponses en provenance de cette FAQ

Comment lister les « sensors » disponibles sur l'appareil ?

Voici un bout de code permettant de lister tous les Sensors disponibles sur le système :

Valide 1.5
```
//Initialisation du manager
SensorManager m_SensorMgr = (SensorManager)
m_Context.getSystemService(Context.SENSOR_SERVICE
);

List<Sensor> listSensor =
m_SensorMgr.getSensorList(Sensor.TYPE_ALL);
for(Sensor sen : listSensor) {
    Log.d("Sensor" ,sen.getName());
}
```

Par exemple si vous souhaitez utiliser l'accéléromètre :

Valide 1.5
```
Sensor m_AcceloSensor =
m_SensorMgr.getDefaultSensor(Sensor.TYPE_ACCELERO
METER);
```

Que sont les Activity et View ?

Une activité est la composante principale d'une application sous Android. L'activity est le métier de l'application et possède généralement une View au minimum, c'est-à-dire un écran graphique. Ainsi dans une application standard, on pourrait trouver une activité qui liste des contacts, une activité qui ajoute un nouveau contact et une activité qui affiche le détail d'un contact. Le tout forme un ensemble cohérent, mais chaque activité pourrait fonctionner de manière autonome.

Que représentent les Intents et les Intent Filters ?

Les Intents permettent de communiquer entre les différentes activités de notre application, mais aussi du téléphone. Ils sont en quelque sorte le "messager" pour lancer une activité. Ainsi une activité peut en lancer une autre soit en passant un Intent vide, soit en lui passant des paramètres. Les Intent Filters jouent le rôle de filtre. Ils permettent de contrôler d'où provient l'Intent (ou d'autres paramètres) afin de lancer ou non l'activité.

Que contient le fichier AndroidManifest.xml ?

Le fichier manifest permet de décrire votre application. On y retrouve :

- le nom du package de l'application. Il servira d'identifiant unique ;
- tous les composants de l'application (Activities, Services, BroadCast Receivers, Content providers). On y décrit également les classes qui implémentent ces composants et leurs capacités (par exemple les Intents qu'elles attendent). Ces déclarations permettent à Android de savoir quels composants sont présents et dans quelles conditions ils s'exécutent ;
- on détermine dans quels processus les composants de l'application seront contenus ;
- les permissions nécessaires pour le bon fonctionnement de l'application ;
- les permissions nécessaires pour que les autres applications utilisent les composants de votre application ;
- les informations contenant les versions de l'Android API requis pour exécuter votre application ;
- les librairies utilisées par votre application.

Qu'est ce qu'un Service ?

Un service ne possède pas d'interface graphique, mais permet de dérouler un algorithme sur un temps indéfini. Il s'arrêtera lorsque sa tâche sera finie ou lorsqu'il sera arrêté. Il peut être, soit exécuté lors du lancement du téléphone (ou tout autre mécanisme interceptable : arrivée d'un appel, d'un sms, etc.), soit au cours d'une action particulière dans votre application via un broadcast receivers.

Que sont les Content Providers ?

Les Content Providers permettent d'accéder à un ensemble de données depuis une application. Vous pouvez ainsi accéder aux contacts, à l'agenda, aux photos et autres données et informations de votre téléphone via des Content Providers. Vous pouvez également définir vos propres Content Providers pour accéder à vos objets, mais également pour que d'autres applications utilisent vos données.

Que sont les Broadcast Receivers ?

Un Broadcast Receiver permet d'écouter ce qui se passe sur le système et éventuellement de déclencher une action si besoin. C'est souvent par ce mécanisme que les services sont lancés.

Comment accéder aux calendriers ?

L'API permettant de manipuler les données du calendrier n'est pas encore ouverte au grand public, il faut donc aller

inspecter un peu dans le code source d'Android afin de pouvoir travailler avec le calendrier.
Ainsi pour récupérer, par exemple, la liste des calendriers de l'utilisateur :

Valide 1.5
```
static public void loadCalendar(Context context){
    ContentResolver contentResolver =
context.getContentResolver();

    // Fetch a list of all calendars synced with
the device, their display names and whether the
    // user has them selected for display.
    inal Cursor cursor =
contentResolver.query(Uri.parse("content://calend
ar/calendars"),
            (new String[] { "_id", "displayName",
"selected","color" }), null, null, null);
    // For a full list of available columns see
http://tinyurl.com/yfbg76w

    ArrayList<PersonnalCalendar> calendarList =
new ArrayList<PersonnalCalendar>();

    while (cursor.moveToNext()) {

        final String _id = cursor.getString(0);
        final String displayName =
cursor.getString(1);
        final Boolean selected = !
cursor.getString(2).equals("0");
        final Integer color = cursor.getInt(3);

        Log.e("LoadCalendar", "Id: " + _id + "
Display Name: " + displayName + " Selected: " +
selected);
    }
}
```

Pour avoir la liste complète des champs utilisables dans le provider du Calendar, rendez-vous à cette adresse : Calendar.java (Lien6)

Comment accéder aux contacts ?

L'accès aux contacts se fait via les Content Providers. Suivant les versions du SDK, il y a plusieurs manières d'accéder à la liste des contacts.

Valide 1.5
```
public static void initContact(Activity
anActivity) {

    //Création de la projection
    String[] phoneProjection = new String[]
{ Contacts.Phones.PERSON_ID,
Contacts.Phones.NAME, Contacts.Phones.NUMBER};

    // Création et initialisation du curseur
    Cursor contactPhoneCursor =
anActivity.getContentResolver().query(Contacts.Ph
ones.CONTENT_URI, phoneProjection, null, null,
null);

    // On laisse l'activity gérer le curseur
    anActivity.startManagingCursor(contactPhoneCu
rsor);

    // On parcourt le curseur
    if (contactPhoneCursor.moveToFirst()) {
        do {
            long personId =
contactPhoneCursor.getLong(contactCursor.getColum
nIndex(Contacts.Phones.PERSON_ID));
            String name =
contactPhoneCursor.getString(contactCursor.getCol
umnIndex(Contacts.Phones.NAME));
            String phone =
contactPhoneCursor.getString(contactCursor.getCol
umnIndex(Contacts.Phones.NUMBER));

            //TODO : faire quelques choses avec
ces informations...

        } while(contactCursor.moveToNext());
    }
}
```

À partir du SDK 2.0, il faut passer par la classe ContactsContract (Lien7). Un exemple d'utilisation se trouve sur le site des développeurs Android : Lien8

Comment faire pivoter l'émulateur ?

Vous pouvez faire pivoter l'écran de l'émulateur en utilisant la combinaison de touche CTRL-F12

Que faire en cas de perte de connexion avec le device ?

Une perte de la connexion entre le device et ADB peut se produire.
Cela se traduit par une impossibilité de piloter le périphérique (émulateur ou vrai téléphone) via le programme ADB (et donc via Eclipse et le plugin DDMS).
Pour pallier à ce problème, il suffit de redémarrer le serveur gérant les transactions. En le tuant d'abord :

```
adb kill-server
```

Puis en le redémarrant :

```
adl start-server
```

Ensuite, pour vérifier la bonne gestion des devices, il suffit de vérifier lesquels sont connectés via la commande :

```
adb devices
```

Que faire si les logs dans la vue LogCat ne s'affichent plus ?

Il arrive de temps en temps que les logs ne s'affichent plus correctement dans Eclipse dans la vue LogCat. Afin de retrouver le comportement normal, allez dans la vue "Devices" et présélectionnez votre device. Ceci devrait résoudre le problème et afficher de nouveau les logs dans la vue dédiée.

Retrouvez la FAQ Android en ligne : Lien9

Les threads composants une application Android

Cet article présente les threads d'une application Android.

1. Introduction

Vous avez probablement déjà réalisé une application Android sans vous soucier du fonctionnement des threads dans celle-ci. Cela est valable peut-être pour certains types d'applications, mais il est important de ne pas négliger cela notamment pour parfaitement contrôler le comportement de votre application.

L'objectif de cet article va être de faire le tour d'horizon des threads créés par une application Android. Le rôle de chaque thread sera évoqué et nous terminerons par la présentation de la création de threads.

2. Les threads d'une application

Toute application Android est composée d'une multitude de threads. Certains vous sont complètement inutiles, d'autres sont plus intéressants et puis il est toujours bon de connaître le fonctionnement global d'Android.

Je propose de baser notre analyse sur une capture d'écran de la perspective **DDMS** d'Eclipse. Cette perspective est ajoutée par le **plug-in ADT** et est dédiée au débogage d'applications. La partie de la perspective qui nous intéresse est "l'afficheur des threads d'une application". En sélectionnant une application au hasard tournant soit sur un Android Virtual Device (AVD), soit sur un téléphone physique, nous pouvons avoir la liste des threads.

Threads | File Explorer | Heap

ID	Tid	Status	utime	stime	Name
3	217	wait	132	54	main
*5	218	vmwait	0	6	HeapWorker
*7	219	vmwait	0	0	Signal Catcher
*9	220	running	6	5	JDWP
11	221	native	0	0	Binder Thread #1
13	222	native	0	0	Binder Thread #2

Les threads d'une application

Nous pouvons constater que cette application utilise six threads. Bien entendu, une grande partie de ces threads ne sont pas créés explicitement par le développeur, certains sont liés à l'utilisation du langage Java et d'autres sont créés par le système d'exploitation Android lui-même. Nous allons lister un à un les threads présents sur cette capture d'écran et décrire brièvement le rôle de chacun d'eux.

- **main** : appelé aussi UI Thread, il exécute l'activity ou le service et s'occupe de l'affichage.
- **HeapWork** : le thread du garbage collector.
- **JDWP (Java debug Wired Protocol)** : thread dédié au débogage (photo prise grâce au débogueur).
- **Signal Catcher** : thread dédié à la réception de signaux ("signal" UNIX).
- **Binder Thread #x** : threads créés par le système (utiles au bon fonctionnement d'Android).

2.1. L'UI thread

Ce thread est le fil de vie de votre Activity ou Service. Ainsi, les instructions rédigées dans les méthodes **onCreate(), onStart(), OnPause(), onResume(), onStop(), onDestroy()** de votre Activity ou les méthodes **onCreate(), onStart(), onStartCommande(), onBind(), onUnbind(), onDestroy()** de votre service sont toutes exécutées dans ce thread.

```
public class TestActivity extends Activity {
    ...
    public void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
        //Code exécuté dans le thread principal
    }
    ...
}
```

Ce thread est également appelé **UI thread** car il est responsable de l'affichage et des interactions avec l'utilisateur. Par exemple, si vous touchez l'écran du doigt, ce thread entre en action et est averti de votre toucher. Son rôle va être de rediriger cette action aux bons éléments de la vue.

L'UI thread est le seul thread qui doit modifier l'affichage. Et pour contourner ce problème, nous pouvons lui soumettre des instructions sous la forme de Runnable (Lien10) grâce notamment à la méthode **runOnUiThread()** de l'Activity.

Ce thread n'est pas fait pour effectuer des **traitements consommateurs en temps**. Tout d'abord si le traitement excède le délai de cinq secondes vous aurez le traditionnel message "application not responsive". Ensuite, en monopolisant le thread vous ne le laissez pas faire son rôle primaire qui est de scruter les actions de l'utilisateur et modifier l'affichage.

Tous les threads peuvent appeler des méthodes modifiant la vue telles que setText() sur un TextView, cependant **le résultat est garanti uniquement s'il s'agit d'un appel effectué depuis l'UI thread**.

2.2. Le garbage collector

Ce thread est la particularité du langage Java, c'est lui qui est chargé d'analyser la mémoire à la recherche de variables qui ne sont plus référencées par votre code dans le but de **supprimer les variables et objets qui ne sont plus utilisés**. Le garbage collector d'Android est assez

simpliste puisqu'il s'agit d'un garbage collector "**mark and swipe**" et non générationnel comme sur les Java Desktop actuels. Pour faire son traitement, le garbage collector bloque l'exécution du programme pendant en moyenne 100 ms. Cette contrainte peut parfois être pénible pour l'utilisateur car elle se caractérise par un gel de l'écran si une animation avait lieu, ou alors une application ne répondant plus du tout, pendant que le garbage collector effectue son ménage. Il est ainsi important de ne pas instancier de trop nombreuses variables ou objets dans des boucles critiques. En effet, la multiplication des allocations peut déclencher un nettoyage de la mémoire par le garbage et donc ralentir l'exécution.

À des fins de test, nous pouvons demander explicitement le nettoyage de la mémoire grâce à l'appel **System.gc()**. Ce dernier permet de constater le temps que peut prendre un nettoyage de la mémoire et notamment tester les effets d'une telle action sur votre application.

2.3. Le thread "Signal Catcher"

Les informations sur ce thread ne concernent que les curieux, car le développeur lambda n'a jamais recours à ce thread.

Comme son nom l'indique, ce thread réagit à des **signaux UNIX** et réalise différentes opérations en fonction du signal reçu. Il peut par exemple stopper l'exécution de tous les autres threads pour faire une copie de la pile, un "dump stack" et une copie de la pile de la JVM Dalvik. Il peut également forcer l'exécution du garbage collector. Ceci est une liste non exhaustive de ses actions. Vous l'aurez compris, son fonctionnement est assez technique et **vous n'aurez pas à vous soucier de l'existence de ce thread**.

2.4. Les "Binder thread"

Tout comme précédemment, le développeur n'aura jamais besoin de manipuler ce type de threads.

Ces threads sont des réservoirs à threads (threads pool). Ils sont **créés par le système Android** dans le but d'optimiser la création de threads notamment par la réutilisation d'anciens threads. Si on crée beaucoup de threads dans une application, on peut observer le nombre de "Binder thread" s'accroître. Qu'on se rassure, leur existence n'implique pas une consommation de temps CPU.

3. Création de threads

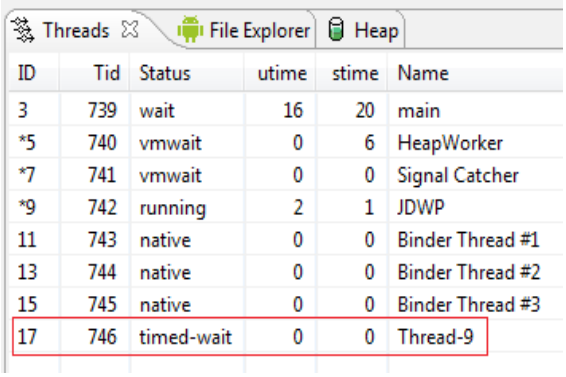
Dans une grande majorité des cas, l'UI thread permet de réaliser toutes les opérations nécessaires au fonctionnement de votre application. Cependant, certaines actions sont consommatrices en temps et peuvent même nécessiter des **opérations asynchrones**. Dans ce cas précis, il est intéressant d'envisager la création de threads. Le thread peut, par exemple, se charger de compléter au fur et à mesure les informations affichées à l'écran. Typiquement, il complète une liste, récupère des informations d'Internet, consulte la base de données des contacts, etc.

La classe incriminée est Thread (Lien11). Généralement, nous l'instancions en lui donnant un **Runnable** qui contient le code qui sera exécuté par le thread fraîchement créé. L'instanciation de l'objet est la première étape et la seconde est l'appel à la méthode **start()**. Cette dernière va créer et démarrer l'exécution du thread. Le thread existera pendant l'exécution du Runnable et mettra fin à ses jours lorsque toutes les instructions auront été exécutées.

```
public class testActivity extends Activity {
        ...
    @Override
    public void onCreate(Bundle
savedInstanceState) {
        //Code exécuté dans le thread principal
        super.onCreate(savedInstanceState);
        ...
        setContentView(R.layout.main);
        new Thread(new Runnable() {
            @Override
            public void run() {
                        //Code exécuté dans le
nouveau thread
            }
        }).start();
    }
    ...
}
```

Pour afficher la liste des contacts, nous pouvons récupérer l'ensemble des noms de ceux-ci et les afficher immédiatement. Ces actions sont réalisées dans l'Activity, alors que, en parallèle, dans un thread fraîchement créé nous pouvons récupérer les photos des contacts et les soumettre à l'UI thread pour qu'il les ajoute à la vue.

Le thread créé peut être visualisé sous la perspective DDMS. Il s'agit exactement de la même vue d'Eclipse que précédemment.



ID	Tid	Status	utime	stime	Name
3	739	wait	16	20	main
*5	740	vmwait	0	6	HeapWorker
*7	741	vmwait	0	0	Signal Catcher
*9	742	running	2	1	JDWP
11	743	native	0	0	Binder Thread #1
13	744	native	0	0	Binder Thread #2
15	745	native	0	0	Binder Thread #3
17	746	timed-wait	0	0	Thread-9

Le nouveau thread visualisé sous DDMS

On peut observer qu'un nouveau "binder thread" a été créé. Et nous pouvons même affirmer que ce "binder thread" est le créateur du thread nouvellement formé.

4. Conclusion

Nous avons présenté les threads composant une application Android. Plus que le nombre de threads, c'est le rôle de ceux-ci qu'il est bon de connaître. La plupart ne sont pas manipulés par le développeur lui-même, mais leurs effets doivent être connus, notamment celui du garbage collector.

Retrouvez l'article de Davy Leggieri en ligne : Lien12

Logiciel de test de Data Warehouse sur la plateforme NetBeans

Maciej Mirski travaille depuis les trois dernières années en tant que testeur pour un centre offshore basé en Pologne. La principale activité de son travail est de tester des Data Warehouse.
En s'axant sur son expérience, Maciej a décidé de créer un outil de test de Data Warehouse quand il travaillait sur sa thèse à l'institut des technologies de l'information Warsaw.

1. L'interview

Bonjour Maciej, peux-tu nous dire quelques mots à propos de l'application sur laquelle tu travailles pour ta thèse.

Bien, J'en suis venu à réaliser qu'il y a un grand fossé entre les outils pour le developpement et les outils pour tester dans le milieu des Data Warehouse. Bien que les développeurs utilisent des outils pratiques comme DataStage ou Informatica, les testeurs utilisent un peu plus de choses que des requêtes SQL. Disposer d'outils qui peuvent aider le modèle et des scénarios de test Retrigger en cas de besoin serait un grand avantage pour les testeurs de Data Warehouse.

L'idée de l'application sur laquelle je travaille actuellement est venue après avoir joué avec CubicTest, une application pour la conception graphique de tests d'interface. Les scénarios créés à l'aide de cet outil peuvent être exécutés en utilisant Watir ou Sélénium. J'ai commencé à penser : pourquoi ne pas disposer d'un outil comme ça pour les essais concernant les Data Warehouse.

Alors, tout d'abord, je voulais une représentation graphique du scénario d'essai, ce qui serait facile à lire et à ajuster. J'ai creusé dans nos tests de cas, ainsi que sur des articles sur les tests de Data Warehouse disponibles sur Internet. Cela m'a aidé à comprendre quels sont les types possibles d'objets / comparaisons de données nécessaires, mais aussi que d'autres types d'actions seraient alors à portée de main pour un testeur.

De cette idée, comment avez-vous développé l'application ?

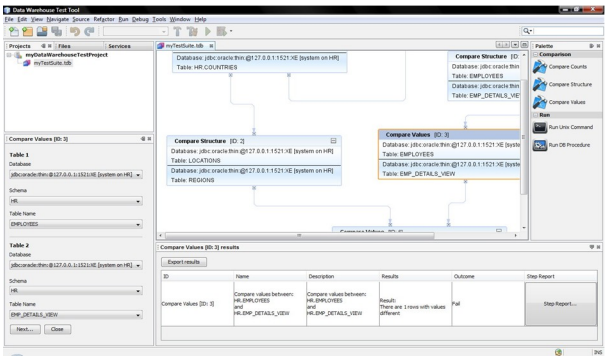
J'ai commencé le développement de la couche bas niveau des étapes de test, sans aucune représentation graphique. Au départ je voulais utiliser Eclipse et son GMF à cet effet, mais après un mois, j'étais encore au même point, à essayer de comprendre comment le tout fonctionnait.

Depuis le temps était compté et j'ai commencé à regarder d'autres bibliothèques que je pourrais utiliser. Après un certain temps, j'ai découvert qu'il y a une autre bibliothèque graphique très pratique, la bibliothèque visuelle qui fait partie de la plate-forme NetBeans. Et aussi parce que ma thèse a été, dès le début, destinée à être un module pour un IDE, je suis passé sur la plate-forme NetBeans.

Après une semaine, j'ai eu une bonne compréhension de la façon de développer quelque chose qui fonctionne via la bibliothèque Visuel. Alors mon développement réel / courbe d'apprentissage a commencé ! Je peux dire qu'il y avait beaucoup de possibilités de jouer avec la plate-forme NetBeans, souvent en ajoutant quelques fonctionnalités seulement parce c'était cool ou je l'ai trouvé par hasard dans la recherche d'autre chose sur les blogs sur la plate-forme NetBeans.

Peut-on voir une capture d'écran où en sont les choses en ce moment ?

Bien sûr.



Ainsi, le principal avantage de la plate-forme NetBeans a été la Bibliothèque visuelle, vous permettant de fournir les widgets vus ci-dessus ?

Oui, la partie principale de mes "besoins" auxquels la plate-forme NetBeans répondait est la bibliothèque Visual. Cependant, il y avait aussi beaucoup d'autres situations où j'ai trouvé quelque chose dans la plate-forme NetBeans que je n'avais pas prévu et que j'ai pu adapter rapidement à ma demande. Par exemple, ce fut le cas avec la base de données API Explorer, que j'utilise actuellement dans mon application.

Pour résumer, voici les API de NetBeans les plus utiles, du point de vue de mes besoins :

o **Bibliothèque Visuelle** : elle est très maniable, a de bons tutoriaux, vous pouvez obtenir jusqu'à la vitesse dans un temps relativement court. Lorsque vous avez besoin de plus hors de lui, il faut creuser plus profond et vous l'avez. Donc, c'est à la fois facile de rattraper son retard et cela vous offre des possibilités vraiment bien quand vous voulez régler.

o **Système de fenêtre** : c'est assez similaire à la bibliothèque de Visual. Il est très facile à utiliser (merci à la manière dont la plate-forme NetBeans gère l'ajout de composants type TopComponents, etc.) et il y a beaucoup d'informations sur Internet lorsque vous avez besoin de changer quelque chose à la manière dont elle fonctionne "hors de la boîte".

o **Database Explorer** : peut-être que ce n'est pas le plus développé dans la plate-forme NetBeans mais j'ai vraiment aimé la façon dont je pouvais facilement l'intégrer à mon module.

Plus généralement, comment voyez-vous le gain général des applications Swing avec la plate-forme NetBeans ?

La plate-forme NetBeans vous donne la possibilité de créer des applications entièrement fonctionnelles. Je pense que dans le monde logiciel d'aujourd'hui, il est nécessaire de s'adapter rapidement et surtout, de réduire les temps de développement en utilisant des bibliothèques déjà disponibles.

Les élèves essaient souvent de créer tout de toutes pièces par eux-mêmes. Je préfère utiliser ce qui est déjà donné, telles que les API de NetBeans (par exemple, la Bibliothèque visuelle) et les combiner avec mon propre code. Cela me donne plus de temps pour vraiment se concentrer sur la mise en oeuvre de mes idées plutôt que sur le code.

Merci à la plate-forme NetBeans, il est également plus facile pour moi de modéliser la façon dont l'utilisateur utilisera mon application. Je pense que sans cela, je ne serais pas allé au-delà que de simplement créer un projet d'étudiant.

Quelle est la situation actuelle de l'outil que vous avez créé et quels sont vos plans pour continuer à le développer ?

Eh bien, je dois dire que ce n'est qu'une thèse de baccalauréat à ce stade, il n'est donc pas un outil très sophistiqué (et j'ai dû apprendre beaucoup de choses sur la plate-forme NetBeans à la volée), mais j'ai l'intention de continuer à le développer.

Retrouvez l'article de Maciej Mirski traduit par Mike Francois en ligne : Lien13

Le développement de PHP6 est suspendu, quand reprendra-t-il ?

Après moult déboires, le développement de PHP6 a finalement été suspendu. L'implémentation d'Unicode dans le futur langage devenait un véritable casse-tête technique.

C'est UTF-16 qui avait été choisi pour le support de l'Unicode dans PHP, mais ce développement s'est avéré difficile et la compatibilité très mauvaise.

Le travail est donc mis en stand-by pour une durée indéterminée, le temps de se vider la tête pour les programmeurs, puis de réfléchir à une autre solution pour l'intégration d'Unicode.

PHP6 sera-t-il achevé un jour ? Certainement.

Mais quand ? Mystère.

Commentez cette news en ligne : Lien14

Sensio Labs présente Symfony 2

C'est le 17 février que Sensio Labs dévoile la version 2.0 du framework PHP Open-Source au cours de la seconde édition du Symfony Live.

Cette version est présentée comme plus flexible, plus maniable, et surtout 3 fois **plus rapide** que la version 1.4 et que ses principaux concurrents ; le tout en **utilisant moitié moins de mémoire** ; c'est surtout **le premier framework php 5.3**.

Symfony 2.0 conserve tous les atouts qui ont fait le succès de Symfony 1.0 :
• la sécurité (XSS, CSRF, SQL Injection...) ;

• la fameuse web debug toolbar notamment ;

• les bonnes pratiques de développement.

Connue pour ses performances mais aussi sa communauté mondiale, Symfony propose une documentation enrichie et une prise en main facilitée, Symfony s'adresse désormais à l'ensemble du monde des développeurs, du débutant au plus expérimenté. Avec seulement une heure d'étude, le développeur serait capable de maitriser les bases du framework.

Attention toutefois, cette nouvelle version n'est pas encore considérée comme stable et ne devrait donc pas être utilisée en production.
À l'heure où ces lignes sont tapées, la version stable de Symfony est la version 1.4.

Commentez cette news en ligne : Lien15

Le CMS open-source Drupal sortira en version hébergée

Courant 2010 pour concurrence WordPress, Google Sites et Microsoft CMS

Acquia, la société qui édite Drupal, vient d'annoncer qu'une version hébergée de son célèbre CMS open-source en PHP – qui sert par exemple à concevoir le site de la NASA – serait lancée vers le milieu de l'année.

Baptisée Drupal Garden, cette version est actuellement en beta-test privé.
Elle sera destinée à concurrencer WordPress, actuel leader sur le marché, et à démocratiser l'outil jusqu'ici assez peu connu du grand public.

Drupal a en effet souvent été critiqué pour sa complexité. Le CMS propose effectivement un grand nombre de modules complémentaires à installer qui le rendent assez peu abordable pour un utilisateur non averti.
Mais ce sont également ces modules qui font toute sa puissance et sa grande adaptabilité aux besoins de ses utilisateurs.

Avec Drupal Garden, Acquia entend grandement simplifier l'utilisation de son outil. Fondé sur le futur Drupal 7, Drupal Garden sera par exemple proposé avec tous les modules de base nécessaires à une utilisation classique installés par défaut.

Ce service grand public sera entièrement gratuit. Deux autres offres, plus complexes, seront proposées elles aux alentours de 20 et 40 dollars. Elles s'adressent aux entreprises désireuses de réaliser des "micro-sites" liés à des gestions de projets, des conférences ou des campagnes de communication qui demandent une conception plus exigeante.

Drupal Garden se positionne donc clairement à la fois en alternative à WordPress et Blogger mais aussi Google Sites et à Microsoft CMS.

Acquia précise par ailleurs que lorsqu'une entreprise décidera de mettre fin à son abonnement, le contenu hébergé sur Drupal Garden ainsi que le code source de l'outil pourront être récupérés pour être installés sur n'importe quel serveur tiers.

Une vidéo de présentation de ce "Drupal 7 as a service" est disponible sur le site de Drupal Garden Beta (Lien16).

Commentez cette news en ligne : Lien17

Gérer le cycle de vie d'une application PHP avec Phing

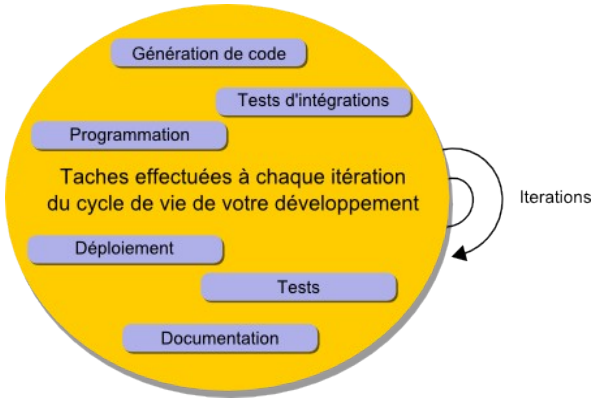
Vous venez de livrer votre produit, un problème apparaît. Vous êtes fatigué, un seul fichier PHP semble incriminé. Vous le corrigez. Vous décidez de passer outre votre procédure de livraison. Vous transférez directement le script PHP "corrigé" sur le serveur de production et là, c'est le drame...

C'est une situation où, aveuglé par la fatigue, l'on s'en veut de s'être comporté comme le pire des débutants. Phing est la solution pour éviter qu'un tel problème ne se reproduise.

Découvrez comment maîtriser chacune des itérations du cycle de vie de votre projet PHP.

1. Présentation

Phing est un outil inspiré d'apache Ant et destiné à PHP. Vous pouvez l'utiliser pour vous assister dans la réalisation de certaines tâches tout au long du cycle de vie (Lien18) de vos projets tels que la génération de code, la construction des releases, la génération de la documentation, la génération de rapports (tests unitaires, couverture du code, aspect du code...), le déploiement dans différents environnements...



Phing vous permet de gagner du temps à chacune de ces étapes

Vous découvrirez, au travers de ce document :

- les avantages de Phing par rapport à un système de build construit à partir de scripts consoles ;
- l'installation de Phing et la mise en œuvre d'un script de build ;
- 2 exemples exécutables pour découvrir les services offerts par Phing ;
- les principaux outils supportés par Phing.

2. Les avantages de Phing

Phing a de nombreux avantages, comparé à un système de build plus archaïque construit à partir d'un ensemble de scripts de déploiement (fichiers PHP, Shell ou batch).

2.1. Il centralise les informations de build dans un seul fichier

Toutes les informations dont Phing a besoin pour fonctionner sont rassemblées dans le fichier build.xml. Vous y décrivez toutes les tâches que vous souhaitez réaliser. Vous pouvez construire des groupes de tâches (appelés targets) réutilisables.

Le fichier build.xml est suffisamment verbeux pour être, lors de sa lecture, compréhensible sans connaissance préalable par un autre développeur .

2.2. Il fonctionne sur toutes les plateformes où PHP est disponible

Phing est codé en PHP. Vous pouvez donc l'utiliser dès lors que PHP est installé sur votre système.

Certaines fonctionnalités avancées peuvent ne pas fonctionner partout (par exemple, CHMOD).

2.3. Il supporte un grand nombre d'outils PHP

Il existe un grand nombre d'applications destinées à améliorer l'écriture du code PHP. Elles ont l'inconvénient de ne pas suivre les mêmes conventions ou la même logique. Phing intègre des tâches dédiées à certaines d'entre elles. Ces tâches sont normalisées et suivent une logique commune. Ce canevas est utile pour paramétrer, efficacement et avec un minimum d'effort, chacune d'entre elles sans sacrifier pour autant les possibilités de réglages avancés.

2.4. Il est documenté

La documentation (Lien19) est d'excellente qualité et détaille avec précision le rôle et la configuration de chaque tâche. Que celle-ci fasse partie du noyau de Phing ou soit rattachée à un outil externe, vous trouverez fréquemment des exemples clairs et concis. Malheureusement, elle n'est pour le moment disponible qu'en anglais.

2.5. Il est extensible

Vous pouvez étendre Phing pour y ajouter de nouvelles tâches. Ainsi vous pourrez utiliser vos propres outils irremplaçables ou non implémentés. Le système d'extension est documenté. Vous trouverez dans la documentation officielle un chapitre entier détaillant les possibilités, proposant des gabarits pour travailler et expliquant pas-à-pas le processus de conception d'une nouvelle tâche.

2.6. Il facilite la mise en place d'un processus d'intégration continue

Il est utilisable avec la plupart des outils dédiés à l'intégration continue : Cruise control (Lien20), Hudson, Xinc...

3. Comment installer Phing

Phing est disponible sous la forme d'un package PEAR. L'installation, la communication avec les applications tierces et la mise à jour de votre environnement est plus facile par ce moyen.

3.1. Installer Phing avec PEAR

Cet article ne couvre pas l'installation de PEAR. Si vous souhaitez vérifier sa présence sur votre ordinateur, rendez-vous sur le manuel de PEAR (Lien21).

Pour installer Phing, tapez dans une console la commande suivante :

Installer Phing avec PEAR
```
pear channel-discover pear.phing.info
pear install phing/phing
```

Pour mettre à jour Phing, tapez la commande suivante :

Mettre à jour Phing avec PEAR
```
pear upgrade phing/phing
```

3.2. Installer Phing sans utiliser PEAR

Cette méthode est déconseillée, car Phing faisant appel à des applications tierces fréquemment installées grâce à PEAR, (par exemple, PHPunit ou PhpDocumentor), l'installation de votre environnement de travail peut s'avérer délicate et souffrir d'un manque de cohésion.

Pour commencer, rendez-vous sur le site de Phing (Lien22) et téléchargez la dernière version stable (la 2.4.0 à la date de rédaction de cet article).

Étapes à suivre pour installer Phing sans PEAR :

1. décompressez la distribution dans un dossier. Par la suite, nous ferons référence au chemin d'accès de ce dossier en temps que PHING_HOME ;
2. ajoutez dans la variable d'environnement PATH (Lien23) le chemin complet PHING_HOME/bin ;
3. créez la variable d'environnement PHING_HOME, sa valeur sera le chemin complet de PHING_HOME ;
4. créez la variable d'environnement PHP_COMMAND, sa valeur sera le chemin complet du répertoire contenant l'exécutable php.exe ;
5. créez la variable d'environnement PHP_CLASSPATH, sa valeur sera le chemin complet du répertoire PHING_HOME/classes ;
6. si vous faites appel à d'autres outils à partir de Phing (PHPUnit...), vous devrez probablement ajouter leur chemin dans PHP_CLASSPATH (séparé par un ;).

À la place de PHP_CLASSPATH, vous pouvez ajouter le chemin complet du répertoire PHING_HOME/classes à la variable PHP include_path.

4. Écrivez votre premier script build.xml pour Phing

Maintenant que Phing est installé, vous êtes prêt à lancer un build.

Ce premier projet est composé de trois fichiers :

- helloworld.php : ce script affiche "Hello world" sur la sortie standard ;
- phpinfo.php : ce script affiche le phpinfo de votre environnement php, utile durant le développement ;
- build.xml : script XML de configuration de Phing.

Votre premier fichier de build copiera le script helloworld.php dans un sous-dossier build/release (qu'il créera si celui-ci n'existe pas).

Les sources de cet exemple sont disponibles dans le package d'exemple (Lien24).

4.1. Les fichiers PHP

Le code de ces fichiers est volontairement simple. L'objectif de cet article n'étant pas d'expliquer comment l'on écrit du PHP.

helloword.php
```
<?php
echo 'hello world';
?>
```

phpinfo.php
```
<?php
phpinfo();
?>
```

4.2. Le fichier build.xml

Voici le fichier que vous allez exécuter. Par la suite, vous trouverez l'explication détaillée de chacun des éléments.

build.xml
```
<?xml version="1.0" encoding="UTF-8"?>

<project name="SimpleHelloworld" basedir="."
default="build">
    <target name="build">
        <echo msg="Debut du build" />

        <echo msg="Création du répertoire
build" />
        <mkdir dir="./build" />

        <echo msg="Création du répertoire
release" />
        <mkdir dir="./build/release" />

        <echo msg="Copie du fichier
helloworld.php vers build/release/helloworld.php"
/>
        <copy file="helloworld.php"
tofile="./build/release/helloworld.php /">
        <echo msg="Fin du build" />
    </target>
</project>
```

4.3. Exécutez l'opération de build

Pour exécuter votre build, vous devez ouvrir une console dans le répertoire du projet. Tapez-y simplement Phing.

Vous devez obtenir une sortie sensiblement équivalente sur votre console

```
Buildfile: D:\Programming\Projects\0019-PhingDiscoveryArticle\svn\trunk\app\01-simple-helloworld\build.xml

SimpleHelloworld > build:

     [echo] Debut du build
     [echo] Creation du repertoire build
    [mkdir] Created dir: D:\Programming\Projects\0019-PhingDiscoveryArticle\svn\trunk\app\01-simple-helloworld\build
     [echo] Creation du repertoire release
    [mkdir] Created dir: D:\Programming\Projects\0019-PhingDiscoveryArticle\svn\trunk\app\01-simple-helloworld\build\release
     [echo] Copie du fichier helloworld.php vers build/release/helloworld.php
     [copy] Copying 1 file to D:\Programming\Projects\0019-PhingDiscoveryArticle\svn\trunk\app\01-simple-helloworld\build\release
     [echo] Fin du build

BUILD FINISHED

Total time: 0.4153 seconds
```

Vérifiez que le fichier helloworld.php a bien été copié dans le sous-dossier build/release.

4.4. L'entête du build

Build.xml commence toujours par la balise project.

Le noeud project

```
<project name="SimpleHelloworld" basedir="." default="build">
```

Les arguments de project :

- name : c'est le nom du projet. Cet argument n'influe pas sur l'exécution de script ;
- basedir : c'est le chemin du répertoire d'exécution du script ;
- default : c'est le nom de la target (cible) visée par défaut, si aucun argument n'est transmis à Phing au moment de l'exécution.

4.5. Les tâches du script

Pour configurer vos tâches, vous les groupez dans une balise target. Ici, elle s'appelle build.

Le noeud target

```
<target name="build">
```

Comme l'instruction echo, la tâche echo affiche un message sur la sortie standard.

La tâche echo

```
<echo msg="Debut du build" />
```

Pour créer un dossier, vous utilisez la tâche mkdir. L'argument dir contient le chemin du dossier à créer. Si le dossier existe déjà, Phing passe à la tâche suivante.

la tâche mkdir

```
<mkdir dir="./build" />
```

Pour copier un fichier, vous utilisez la tâche copy. L'argument file contient le chemin du fichier dont vous souhaitez copier le contenu et l'argument tofile le chemin du fichier où vous souhaitez coller le contenu.

la tâche copy

```
<copy file="helloworld.php" tofile="./build/release/helloworld.php /">
```

5. Les autres exemples

Les deux exemples supplémentaires, disponibles dans le package d'exemples (Lien24), vous donneront un meilleur aperçu du potentiel de Phing. Je ne les détaillerai pas. Je vous encourage à les exécuter et à ouvrir le fichier build.xml pour chacun d'eux.

Pour les utilisateurs de Windows, chacun des projets contient un fichier cmd.bat permettant d'ouvrir une console dans le dossier courant.

5.1. 02-helloworld-package-release-by-version-in-zip

Cet exemple illustre l'utilisation de propriétés pour configurer le build et la compression au format ZIP.

README.dev.txt

```
Qu'est-ce que c'est ?
---------------------

Vous pourrez grâce à cet exemple exécuter un script de build qui archive
automatiquement chacune de vos releases. Ce build est adapté aux petits
projets pour lesquels vous travaillez souvent seul et pour lesquels vous n'employez
pas de serveur de version.

Le fichier build.xml propose les opérations suivantes :
 # déploiement de l'application dans l'environnement de debug ;
 # déploiement de l'application dans l'environnement de release et création
    d'un package dont le nom est basé sur le numéro de version (si un package
    existe, il ne sera pas remplacé et le script affichera un message d'erreur) ;
 # déploiement forcé de l'application dans l'environnement de release et
    création / remplacement d'un package dont le nom est basé sur le numéro de version.

Pour changer le numéro de version, vous devez éditer la propriété projectVersion
dans le fichier build.xml.

Pour un simple helloworld, ce déploiement est trop lourd, mais vous pouvez l'adapter
à la majorité de vos applications. Les prérequis sont peu nombreux.

Contrairement au build précédent, vous ne configurez pas la copie fichier par fichier
mais vous configurez la tâche de copy pour copier tous les fichiers respectant certaines
conventions. Si vous rajoutez un fichier ou
```

```
plusieurs fichiers à votre application,
vous n'aurez pas à modifier votre script de
build.

Quelles sont les fonctionnalités de Phing
utilisées ?
-------------------------------------------------
-

Ce sont :
# AvailableTask : vérifier qu'un dossier ou un
fichier est disponible ;
# CopyTask : copier un ou plusieurs dossiers /
fichiers ;
# DeleteTask : supprimer un ou plusieurs dossiers
/ fichiers ;
# ExitTask (FailTask) : arrêter l'exécution d'un
script ;
# Fileset : permet de définir un groupe de
fichiers à partir de règles d'inclusion
    et d'exclusion pour les instructions
manipulant des fichiers (CopyTask, DeleteTask...)
;
# IfTask : prise de décision durant le build ;
# Mapper : permet d'effectuer certaines
transformations sur le nom des fichiers
    dans le cadre de certaines opérations
(CopyTask, DeleteTask...) ;
# PropertyTask : utiliser des variables
permettant de modifier
    la configuration du build ;
# Target : utiliser des targets dépendant les
unes des autres ;
# ZipTask : compresser dans un seul fichier à
l'aide de l'algorithme Zip, un ou
    plusieurs dossiers / fichiers ;
```

5.2. 03-helloworld-unittest-codesniffer-phpdocumentor

Pour cet exemple, vous devrez installer PHPUnit, Php_CodeSniffer et PhpDocumentor dans votre environnement PHP. Vous y découvrirez comment utiliser la date pour packager une release.

README.dev.txt

```
Qu'est-ce que c'est ?
---------------------

Cet exemple présente un build plus complexe.
Celui-ci exécute les tests unitaires
PHPUnit, vérife que le code respecte la
convention de code du projet PEAR
grâce à PHP_CodeSniffer et génère la
documentation de l'API en utilisant PhpDoc.

Comme dans les projets précédents, nous allons
sortir un package au format zip
incluant le numéro de version, la date de
l'exécution et le statut des tests
unitaires qui ont été exécutés.

Le nom du package se présentera de la façon
suivante :
NomDuProjet-Version-YYYY-MM-DD-hh-mm-ss-
TestsResult

Pour ce type de projet, l'emploi d'un serveur de
version serait très avantageux.
Cependant, nous allons volontairement l'omettre
dans cet exemple pour des raisons
de facilité de mise en oeuvre.
```

```
Note : dans la pratique, si vous n'avez pas de
serveur de version, vous
pouvez figer votre environnement dans une
archive, en excluant le dossier contenant
tous les packages de release. Ainsi, vous pourrez
en quelques clics revenir à une
version différente de votre développement. Vous
pourrez également utiliser un outil
tel que winmerge pour comparer les deux versions
de votre projet. Il ne s'agit toutefois que
d'un palliatif valable si vous travaillez seul,
un serveur de version restant
recommandé car il rend beaucoup plus de services
que le simple archivage.

Dans cet exemple, vous découvrirez une méthode
pour sauvegarder dans un fichier
séparé le code commun à chacun de vos builds
(procédé recommandé dans le cadre de
l'industrialisation de vos projets).

Le fichier build.xml propose les opérations
suivantes :
 # exécution des tests PHPUnit ;
 # exécution des tests PHP_CodeSniffer (le
fichier app/function.php déclenche deux
    warnings) ;
 # génération de la documentation d'API ;
 # exécution de toutes les opérations de "quality
assurance" (documentation, tests) ;
 # déploiement de l'application dans
l'environnement de debug ;
 # déploiement de l'application dans
l'environnement de release ;
 # packaging de l'application au format ZIP.

Quelles sont les fonctionnalités de Phing
utilisées ?
-------------------------------------------------
-

Ce sont :
# PhingTask : vous pouvez appeler une target dans
un autre fichier de build
    (ceci est utile pour mettre en commun le code
de vos builds) ;
# PhpUnitTask : vous exécutez les tests unitaires
grâce à cette tâche (mais ceci n'est
    pas compatible avec les suites de tests ;
Phing construit ses propres suites) ;
# PhpCodeSnifferTask : vous vérifiez avec cette
tâche le style de votre code
    grâce à PHP_CodeSniffer ;
# PhpDocumentorTask :  vous pouvez générer la
documentation de l'API de votre
    code et le manuel utilisateur à partir de
fichiers docbook grâce à Phpdoc.
```

6. Survol des outils utilisables avec Phing

Dans le chapitre précédent, vous avez découvert l'aspect fonctionnel. Ici vous allez découvrir les outils tiers utilisables dans Phing. Ceux-ci sont souvent écrits en PHP. Le tableau suivant présente une liste non exhaustive des outils supportés par la version 2.4.0.

Voir tableau sur l'article en ligne (Lien25)

Vous trouverez toutes les informations pour utiliser ces outils dans le manuel de Phing (Lien19).

Retrouvez l'article de Fabien Arcellier en ligne : Lien25

Quake 2 en javascript grâce à HTML5

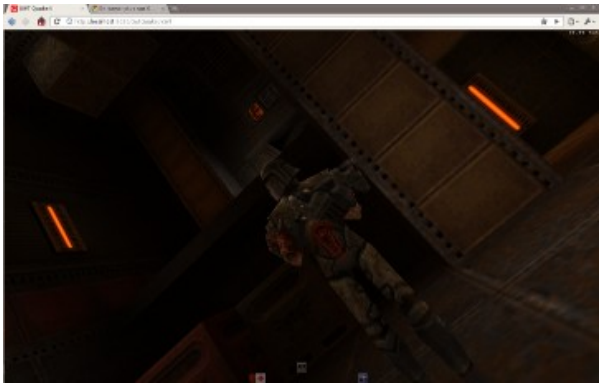
J'ai d'abord cru à un bon gros poisson d'avril bien monté, mais après avoir testé, il n'en est rien. Des cinglés chez google ont bel et bien porté Quake II en HTML5. De quoi en mettre, à mon avis, plein les dents à ceux qui prétendent que HTML5 ne pourra pas concurrencer Flash.

C'est cette fin d'avant midi que mon collègue m'a dit "alors, t'as testé quake II en HTML5 ?", me montrant une vidéo, que vous trouverez ici (Lien26), datée du premier avril et on a bien rigolé! Forcément, qui goberait ça ? Ils prétendent, en utilisant GWT (permettant de faire du html/javascript en java), WebGL (présent dans HTML5) et toute une série d'autres choses, arriver à porter en html la version java existante de Quake II. La bonne blague !
Mais en fouinant un peu, on trouve le projet Google Code associé. Ok, ils ont vraiment bien corsé leur poisson d'avril. Curieux comme je suis, j'ai donc suivi les instructions de téléchargement, compilé le projet avec Maven, je l'ai lancé et voilà le résultat console :

```
/tmp/quake2-gwt-port$ ./run-dedicated-server 6060
2010-04-02 11:12:45.527::INFO:  Logging to STDERR
via org.mortbay.log.StdErrLog
2010-04-02 11:12:45.592::INFO:  jetty-6.1.x
2010-04-02 11:12:45.630::INFO:  Started
SocketConnector@0.0.0.0:8080
couldn't exec config.cfg
couldn't exec config.cfg
ServerWebSocketImpl(27910)
2010-04-02 11:12:46.011:INFO::Logging to
StdErrLog::DEBUG=false via
org.eclipse.jetty.util.log.StdErrLog
Starting Server
2010-04-02 11:12:46.013:INFO::jetty-
7.0.1.v20091125
2010-04-02 11:12:46.067:INFO::Started
SelectChannelConnector@0.0.0.0:27910
Server started
ServerWebSocketImpl(27901)
Starting Server
2010-04-02 11:12:46.069:INFO::jetty-
7.0.1.v20091125
2010-04-02 11:12:46.070:INFO::Started
SelectChannelConnector@0.0.0.0:27901
Server started
==== ShutdownGame ====
==== InitGame ====
------- Server Initialization -------
====== Quake2 Initialized ======

nextmap: demo2
```

Ok, bon ça affiche du beau code dans la console du serveur, bien bien. Je télécharge donc, comme indiqué, la dernière nightly build de Chrome pour Linux (d'après eux ça marche aussi avec le dernier build de Webkit/Safari sous Mac OS X). Je lance le navigateur et le pointe vers l'URL indiquée, me demandant quand le poisson va apparaître en gros à l'écran (tout en me disant qu'un millier de classes java pour afficher un poisson, c'est pousser le vice, là). J'obtiens une jolie console de démarrage QuakeII animée. Jusque-là rien d'interactif, le bon vieux menu "nouvelle partie, multiplayer, demo, load, save, options". Je lance direct une partie et... Merde, ça marche. Je joue, je teste, ce n'est pas une démo mais un vrai jeu qui tourne. Vous pouvez même profiter de ma mort au bout de 10 minutes...



Grâce au navigateur Chrome ou Safari, vous pourrez profiter d'un moteur Quake II tournant dans votre navigateur. Ma machine de bureau, non conçue pour les jeux, arrive à faire tourner celui-ci à 30 FPS en plein écran. Le rendu WebGL est performant, le jeu est fluide, les IA réactives, on est vraiment dans un bon vieux Quake II bel et bien porté. Seul hic, sous Linux je n'ai pas de son (normalement on est censé en avoir).
Pour tous ceux qui veulent tester : Lien27

Pour télécharger et compiler, tapez dans la console :

```
hg clone https://quake2-gwt-
port.googlecode.com/hg/ quake2-gwt-port
```

assurez-vous d'avoir une version java récente installée ainsi que Maven et suivez les instructions (Lien28).

Retrouvez ce billet sur le blog de David Delbecq : Lien29

Les dernières news

Internet Explorer 9 disponible en préversion, Microsoft attend vos retours

En direct du Mix à Las Vegas, qui se passe la nuit pour nous, Microsoft vient d'annoncer la mise en ligne d'une préversion d'Internet Explorer 9. Cette version test permettra d'essayer les nouvelles fonctionnalités du navigateur, mais elle n'est pas adaptée au grand public (par exemple, absence de barre d'adresse, il faut saisir les URL via le menu Page).

En général, Microsoft délivre ce type de produits lorsque le développement en est à un stade plus avancé, mais il est clair que cette fois, exception est faite et que l'éditeur espère bien s'appuyer sur le feedback des utilisateurs pour améliorer au maximum son logiciel.

```
"At this time, we're looking for developer
feedback on our implementation of HTML5's parsing
rules, Selection APIs, XHTML support, and inline
SVG. Within CSS3, we're looking for developer
feedback on IE9's support for Selectors,
Namespaces, Colors, Values, Backgrounds and
Borders, and Fonts. Within DOM, we're looking for
developer feedback on IE9's support for Core,
Events, Style, and Range."
```

a déclaré Dean Hachamovitch, general IE manager chez Microsoft.

D'autres préversions sont attendues à quelques semaines d'intervalle, avant que n'arrive une version bêta puis une RC. Même si IE9 est encore loin d'être achevé et prêt à être commercialisé, il est très prometteur.

Commentez cette news en ligne : Lien30

Google lance le service de Remarketing qui permet de suivre l'internaute partout à travers le Web

Google a annoncé sur le blog officiel d'AdWords le lancement de Remarketing, une nouvelle fonctionnalité pour les annonceurs AdWords, qui permettra à ceux-ci de suivre les internautes sur les différents sites qu'ils consultent sur le Net.

Le système AdSense Classique permet d'afficher des annonces ciblées en fonction du contexte des sites visités, la fonctionnalité Remarketing permet d'ajouter des cookies sur la machine de l'internaute afin de le suivre à travers le Web pour réafficher les mêmes annonces sur les différents sites qu'il consulte, et qui apparient bien évidement au réseau publicitaire de Google.

En d'autres termes cette fonctionnalité permettra aux annonceurs d'afficher les publicités, non seulement selon le contexte de la navigation, mais aussi selon le centre d'intérêt de l'internaute.

Pour clarifier plus le fonctionnement, Google donne l'exemple d'une agence de voyage qui vend des séjours dans les Caraïbes pendant la période de vacances. Plusieurs internautes peuvent réserver des séjours sur le site de l'agence, mais il se peut que plusieurs d'entre eux décident à la dernière minute d'annuler la réservation à cause des prix jugés trop chers par exemple. La fonctionnalité Remarketing va permettre à cette agence de voyage d'atteindre à nouveau plus facilement ces utilisateurs, qui ont déjà exprimé un intérêt pour les voyages dans les Caraïbes, dans le cas d'une baisse de prix par exemple, vu que leurs machines contiennent déjà des cookies qui montrent leurs centres d'intérêt.

La question de la protection de la vie privée se pose à nouveau, vu que cette fonctionnalité se base principalement sur les préférences des utilisateurs, mais Google propose une page Web (Lien31) qui permettra à ceux-ci d'ajouter/supprimer des centres d'intérêt ou de désactiver totalement ces cookies.

Commentez cette news en ligne : Lien32

Les gagnants du Challenge Mappy veulent rendre leur API open-source, interview exclusive des développeurs primés

En février dernier, un concours gratuit et ouvert à tous avait été lancé par Mappy, en partenariat avec Developpez.Com. Il consistait à donner carte blanche aux participants qui devaient créer l'application de leur choix en utilisant les API Mappy AJAX et AS3.

Vendredi 9 avril avait lieu à Paris la remise des prix clôturant ce challenge. Les douze membres du Jury (dont Didier Mouronval, de Developpez) ont été très satisfaits des projets présentés et il ne leur fut pas facile de sélectionner les vainqueurs.

Cinq réalisations furent primées. Présente lors de cette soirée d'annonce officielle du palmarès, j'ai pu interviewer le trio de tête.

Commençons par la médaille de bronze, la troisième place du podium. Ce prix est revenu à Sébastien DUGUÉ, ancien chef de produit chez un éditeur logiciel actuellement à la recherche d'un emploi, qui a entendu parler du challenge par un blog.

Ce professionnel qui se dit "développeur en herbe", a conçu son application fin 2009 avant de la lancer en janvier 2010. Quand il a eu vent du challenge, il a décidé de mettre à jour son travail en y ajoutant plusieurs fonctionnalités (comme l'accès aux ordinateurs via un site, les statistiques, différentes vues cartographiques, etc.). De plus, il a abandonné GoogleMap pour le remplacer par Mappy, dont il décrit l'API comme "très agréable pour développer, au même niveau que Google pour les fonctionnalités mais avec un plus : on peut clustériser les marqueurs sur une carte et les regrouper en un seul. Et puis la simplicité pour créer des formes géométriques sur la carte... C'est bien plus aisé qu'avec Google", argumente-t-il.

Sébastien a mené son projet seul comme un chef. Il faut dire que lorsqu'il travaillait en tant que CRM, il coordonnais le développement des logiciels et en rédigeait le cahier des charges. Des compétences qui lui ont servit pour mener à bien ce challenge.

iBordeaux fonctionne sur tous les smartphones et a été programmée avec PHP, MySQL et Javascript.

La même application existe pour la ville de Rennes, sous le nom d'iRennes. Sébastien aimerait maintenant proposer son idée à d'autres municipalités pour une déclinaison de son concept. Des applications clés en mains, pour des villes ou des organismes de transport. L'idée : faciliter les transports citadins, avec des plans, les horaires de passage

du prochain tramway ou métro, des plans affichant les vélos en libre service, etc.

Vous pouvez essayer ses applications sur leur site officiel : Lien33 (celle primée par le challenge) et Lien34

Sébastien et son prix

Continuons avec la médaille d'argent, la deuxième place du podium. Vincent DE OLIVEIRA et Cedric ESNAULT sont deux professeurs d'informatique qui enseignent à l'école nationale des sciences géographiques (intégrée à l'IGN). Collègues, ils sont habitués à collaborer. Aussi, quand l'un d'eux découvre l'existence du concours sur le site Internet de l'IGN, c'est tout naturellement que les deux hommes s'associent pour y participer.

Ce sera la première fois qu'ils développeront une API ensemble. Vincent nous indique que l'idée précise de WIMT lui est venue alors qu'il se rendait vers une gare de RER en voiture : "Ça serait super d'avoir les informations en temps réel et de manière visuelle, plutôt que via une liste d'horaires".

Pendant un mois et sept jours, les deux enseignants ont travaillé sur leur projet le soir après les cours. Chacun de leur côté, mettant les données en commun par des mises à jour à distance. "Le développement s'est fait avec ASVN", explique Cedric. "On a créé une application pour parser automatiquement les horaires de la RATP en Java, depuis les PDF d'origine. Notre base de données était ensuite directement remplie avec des informations", poursuit-il.

WIMT est capable de gèrer les horaires à la seconde près, malheureusement les informations récupérées sur le site Internet de la RATP ne mentionnent que les temps à la minute. "Du coup, il arrivait qu'un train se retrouve dans deux gares proches en même temps", raconte Vincent.

Il a fallu aux deux développeurs faire preuve d'astuce pour interpoler les temps de passage aux gares. Leur application Java remplissait les vides (par exemple, certains trains ne s'arrêtent pas à toutes les stations, le PDF ne mentionne donc pas l'heure à laquelle ils y passent), mais ce souci de temps arrondi à la minute leur a demandé beaucoup de travail de correction.

Les deux jeunes gens espèrent aujourd'hui obtenir des informations supplémentaires de la part de la RATP : les horaires à la seconde près, mais aussi les retards, les accidents, etc... "C'est techniquement faisable avec l'API de Mappy, il ne manque plus que sa collaboration et celle de la SNCF aussi pour d'autres lignes. WIMT peut s'étendre à d'autres modes de transport : métro, trains, bus, etc...".

Pour l'instant, l'application ne supporte pas la charge d'un grand nombre de visiteurs, mais cela pourrait changer.

Elle est également disponible en version allégée pour les smartphones (sous Android, iphone, etc... L'OS est automatiquement détecté).

Pour en savoir plus, visitez le site officiel de l'application : Lien35

Cédric et Vincent tenant leur prix

Terminons avec les champions de la soirée, en première position : Séverine et Raphaël CANDELIER. Issus de la même fratrie, tous deux se passionnent pour les nouvelles technologies. Lui, 28 ans, est professeur de Physique et webmaster du site de la Societé Française de Physique. Elle, 32 ans, monte actuellement une société et un site de covoiturage (le service sera opérationnel dans deux ou trois semaines). Dans le cadre de son projet, il lui fallait une API géographique. Elle a porté son choix sur celle de Mappy qui "contrairement à celle de Google, calcule les péages". Elle a donc intégré l'API de Mappy sur son site et c'est ainsi qu'elle a eu vent du concours.

Le frère et la soeur ont alors réfléchi : "On voulait mettre en valeur les bases géographiques. Comme Raphaël est un grand amateur de jeux, l'idée est venue d'ajouter un aspect dynamique au développement".

Le développement à demandé deux mois de travail acharné. "Mappy fonctionne par couches, il y a une couche pour chaque chose. On a crée une couche supplémentaire qui communique avec les autres et que l'on contrôle. Cette couche comprend des marqueurs mobiles qui se déplacent sur les cartes en suivant les itinéraires calculés par Mappy. On utilise les outils de Mappy pour les faire bouger : une requête est envoyée à Mappy pour calculer l'itinéraire, on le récupère sous forme de segments qui représentent les routes et le marqueur se déplace en les suivants.", approfondit Séverine.

Transformer Mappy en jeu, c'est original. Et c'est l'un des aspect de ce projet qui a séduit le jury. Avant vendredi, l'application était tenue secrète. Elle va désormais être référencée et diffusée, ce qui permettra de tester la résistance à la charge.

Raphaël et Séverine annoncent vouloir la rendre open-source, pour que d'autres "prennent le relais ou utilisent le code".

Vous pouvez découvrir ce jeu, dont le but est d'éviter les monstres qui circulent dans les rues, sur leur site officiel : Lien36

Raphaël et Séverine ont remporté le Grand Prix

Commentez cette news en ligne : Lien37

Les attaques sur le web 2.0 ont augmenté de 500 %, tour d'horizon des dernières tendances cybercriminelles

Mardi, le réseau de distribution d'applications Blue Coat Systems a rendu public un rapport très complet sur les dernières tendances en matière de cybercriminalité.

Ce qui a changé récemment, c'est l'utilisation du Web 2.0 qui s'est envolée en quelques mois. De 2008 à 2009, le nombre d'attaques utilisant des sites communautaires à augmenté de 500 % !

Facebook et Twitter sont en tête de liste des réseaux sociaux visés.

Cette étude analyse les chiffres de 2009.

Le constat général est que, depuis l'avalanche "ILoveYou", les gens ont pris conscience du danger potentiel contenu dans les courriers électroniques. Ils sont donc plus prudents à l'égard de leurs e-mails.

Cependant, beaucoup d'entre eux ne font pas encore le lien entre forums, pages facebook, tweets et menaces. Ils ne réalisent pas que ces autres types de messages numériques peuvent comporter autant de risques que les pièces jointes à un courriel.

Mais le manque de méfiance des internautes envers ces communautés virtuelles n'est pas la seule chose à y attirer les pirates.

Les sites de réseaux sociaux, c'est aussi une manne de victimes potentielles pour les cybercriminels. Des tas de contacts, dont certains (environ 10%) n'utilisent plus l'e-mail mais uniquement les sites communautaires pour communiquer.

Suivant les tendances des internautes, les pirates ont donc investi le web 2.0 : Facebook, Twitter, mais aussi les web apps de Google et les applications pour smartphones. Les marchés en ligne de type AppStore ou Android Market sont une terre fertile pour les cybercriminels. On y trouve déjà des malwares déguisés en jeux.

D'où l'importance que les applications soit vérifiées avant d'être distribuées.

Sur Facebook, l'un des dangers majeurs vient dans le fait d'accepter d'être "ami" avec un total inconnu, qui peut ensuite vous voler des informations personnelles. D'après une enquête menée en décembre 2009 par Sophos, près de 46 % des membres de Facebook accepteraient ainsi l'amitié d'étrangers.

Le succès des attaques sur le web 2.0 viendrait avant tout de la porte ouverte par les victimes, qui ne seraient pas assez prudentes.

Mais les crimes en ligne deviennent aussi de plus en plus sophistiqués. Loin de l'image d'épinal du hacker solitaire isolé dans sa chambre, on se trouve désormais confrontés à du crime organisé. Le "poids" mondial des fraudes informatiques dépasserait le milliard de dollars selon certaines estimations.

Les pirates ont même des outils de statistiques ! Un pack d'analyse du traffic web est par exemple inclus dans le trojan Zeus (qui s'achète au noir).

Les faux antivirus et les faux codecs vidéos restent les malwares les plus utilisés.

Blue Coat recommande aux entreprises de s'équiper d'antivirus en temps réel et de filtres en 2010 ; l'emploi d'un service cloud capable de répondre en temps réel et sans mises à jour manuelles est également préconisé.

L'accent doit également être mis sur les mobiles et les travailleurs en mouvement, qui souvent n'utilisent pas le réseau de l'entreprise.

Commentez cette news en ligne : Lien38

Réaliser un slider facilement grâce au plug-in jQuery Coda-Slider

Le framework jQuery de JavaScript est un outil puissant qui a su s'imposer comme étant un framework de référence pour le développement Web.

1. Introduction

Ce *plug-in* jQuery, sous licence MIT et GPL, a été conçu par Niall Doherty et est fortement inspiré du *slider* Panic Cola.
Il permet de présenter n'importe quel contenu HTML sous la forme d'un *slider*, qui est bien entendu entièrement modifiable en éditant le fichier CSS.

Voici le résultat que vous devriez obtenir très facilement à la suite de cet article : Lien39

Les nouveautés de la V 2.0 :

- défilement automatique des *slides* possible ;
- 1er onglet affiché paramétrable ;
- vitesse de défilement et animation ajustable.

De plus, les liens vers les différents *slides* sont entièrement "*SEO Friendly*", c'est-à-dire qu'ils sont optimisés pour le référencement.
Ce plug-in nécessite trois fichiers JavaScript pour fonctionner : jQuery (fonctionne avec la dernière version de jQuery 1.4), le plug-in easing-jquery ainsi que le plug-in coda-slide lui-même.
Il est compatible avec tous les navigateurs pour peu que le JavaScript soit activé !

2. Installation de Coda-Slider

L'installation du *plug-in* est relativement simple, il suffit de :

- télécharger cette archive (Lien40) et de l'extraire (avec Winrar ou Winzip par exemple). Elle contient tous les fichiers nécessaires ainsi qu'un exemple de *slider* qui pourra vous servir de modèle ;
- intégrer la feuille de style demo-menu.css dans votre page contenant le *slider* (que vous pouvez renommer ou incorporer à votre feuille de style). C'est via cette page CSS que vous pouvez modifier l'aspect du *slider*. Mais attention, ne changez pas le nom des .class ;
- intégrer les trois scripts JavaScript en respectant l'ordre suivant : jQuery, easing et coda-slide.

Voici le code de base qu'il vous faut avant l'intégration du *slider* :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="fr" lang="fr">

  <head>
    <meta http-equiv="Content-type"
content="text/html; charset=UTF-8" />
    <title>Coda-Slider 2.0</title>
    <meta http-equiv="Content-Language"
content="fr" />

    <!-- Link du style CSS -->
    <link rel="stylesheet" href="coda-slider-
2.0.css" type="text/css" media="screen" />

    <!-- Appel des différentes bibliothéques,
attention à l'ordre !-->
    <script type="text/javascript"
src="js/jquery-1.4.js"></script>
    <script type="text/javascript"
src="js/jquery.easing.1.3.js"></script>
    <script type="text/javascript"
src="js/jquery.coda-slider-2.0.js"></script>

    <script type="text/javascript">
      // Vous placerez le code JS ici
    </script>
  </head>

  <body>
<!-- Vous placerez le code HTML du slider ici -->
  </body>

</html>
```

Il ne vous reste plus qu'à y intégrer le code HTML et JavaScript de votre *slider*, ce que nous allons voir maintenant.

3. Le code HTML

Voici le code HTML de notre *slider* :

```
<div class="coda-slider-wrapper">
  <div class="coda-slider preload" id="coda-
slider"> <!-- L'id de ce div est important pour
js-->

    <div class="panel"><!-- Slide 1-->
      <div class="panel-wrapper">
        <h2 class="title">Panel 1</h2>
        <p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. <br/>
          Nam et lacus neque. Sed volutpat ante
id mauris laoreet vestibulum. <br/>
          Morbi vel enim dignissim massa
dignissim commodo vitae quis tellus. <br/>
          Ut enim massa, sodales tempor
convallis et, iaculis ac massa.<br/>
          Aliquam sit amet purus lectus.
Maecenas tempor ornare sollicitudin.</p>
      </div>
    </div>
```

```
    <div class="panel"><!-- Slide 2 -->
      <div class="panel-wrapper">
        <h2 class="title">Panel 2</h2>
          <p>Proin nec turpis eget dolor dictum
lacinia. <br/>
             Suspendisse laoreet ornare
ullamcorper. Nulla in tortor nibh. </p>
      </div>
    </div>

    <div class="panel"><!-- Slide 3 -->
      <div class="panel-wrapper">
        <h2 class="title">Pannel 3</h2>
          <p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. <br/>
             Nam et lacus neque. Sed volutpat
ante id mauris laoreet vestibulum. <br/>
             Morbi vel enim dignissim massa
dignissim commodo vitae quis tellus. <br/>
             Ut enim massa, sodales tempor
convallis et, iaculis ac massa.<br/>
             Aliquam sit amet purus lectus.
Maecenas tempor ornare sollicitudin.</p>
          <p>Proin nec turpis eget dolor dictum
lacinia. <br/>
             Suspendisse laoreet ornare
ullamcorper. Nulla in tortor nibh. </p>
      </div>
    </div>

    <!-- Rajoutez autant de slides que vous le
voulez -->

  </div>
</div>
```

Quelques précisions sur le code :

- ne renommez pas les class, sinon le *plug-in* ne marchera plus ;
- comme vous pouvez le remarquer, il n'y pas de div pour les boutons suivant et précédent. En fait, ils sont créés directement en JavaScript via le *plug-in* ;
- la class 'title' correspond au nom du bouton pour aller à ce slide ;
- le contenu du *slide* peut aussi bien être une image que du texte, voire les deux. Peu importe le nombre de lignes de HTML, la hauteur du slide s'adapte (vous pouvez désactiver cette option).

Vous pouvez voir que le code HTML est relativement simple, mais le JavaScript, quant à lui, l'est encore plus !

4. Le code JavaScript

Il est à placer dans la partie prévue à cet effet comme indiqué dans le code de base.
Nous allons configurer notre *slider* de façon à ce que les *slides* changent automatiquement toutes les X secondes. Un clic sur le *slide* mettra en pause le défilement.

Vous allez voir que ceci est extrêmement facile grâce au plug-in coda-slide !

```
$().ready(function() {
  $('#coda-slider').codaSlider({
    autoSlide: true, // Active l'autoslide
    autoSlideInterval: 4000, // Temps en
millisecondes entre chaque changement
    autoSlideStopWhenClicked: true, // Avec la
pause lors du clic

    autoHeight: true // Mettre à False pour
enlever la fonction qui ajuste la hauteur du
slide
  });
});
```

Le nom des différentes options est très explicite.
Il en existe bien entendu pas mal d'autres. Elles vous sont toutes expliquées dans la liste exhaustive suivante.

5. Les différentes options

Les options de Coda-Slider (voir tableau dans l'article en ligne)

6. Liens externes

- Site officiel : Lien41
- Démonstration : Lien42
- Options disponible du plug-in sur le forum : Lien43

Retrouvez l'article de Julien Itard en ligne : Lien44

Présentation de la programmation logique avec Castor

Cet article a pour but de vous introduire les grands concepts de la programmation logique puis sa mise en pratique en C++ : Castor. Le lecteur n'est pas obligé d'avoir des bases en programmation logique, seules quelques bases en C++ sont nécessaires.

1. Introduction

Cet article est un tutoriel d'introduction à la programmation logique (PL) en C++. Aucune connaissance préalable n'est requise dans des langages qui supportent nativement la programmation logique. Cet article montre aussi comment la PL se lie avec les autres paradigmes supportés par le C++ et avec la STL. La capacité à choisir un paradigme approprié ou un mélange de paradigmes pour résoudre un problème donné est fondamentale et le coeur de la programmation multi-paradigme. Nous commencerons par une brève introduction à la PL, suivie par une discussion de la programmation logique en C++ pour finir avec des exemples. Le support de la programmation logique est fourni par Castor, une bibliothèque Open Source disponible sur www.mpprogramming.com (Lien45). Elle ne nécessite pas d'extension au langage pour compiler les codes sources fournis ici.

2. La programmation logique

La programmation logique est un modèle Turing complet (NdT: cela veut dire qu'on peut coder avec tout ce qu'on peut imaginer) La façon de programmer en utilisant le paradigme logique diffère fondamentalement de la manière utilisée avec la programmation impérative ou fonctionnelle. Des programmes se basant uniquement sur la PL sont entièrement déclaratifs. Quand on programme avec des langages basés sur un paradigme impératif (comme C, C++, Java, ...), les programmeurs dictent à l'ordinateur comment résoudre le problème et l'ordinateur n'a pas connaissance du problème en lui même. Ainsi, les algorithmes jouent un rôle central dans ce modèle de programmation.

Dans des langages de programmation logique tel Prolog ou Gödel, c'est exactement l'inverse qui se produit. En PL, le rôle du programmeur est de fournir des informations sur le problème à l'ordinateur et non de fournir un moyen de résoudre le problème. C'est l'ordinateur qui va appliquer un algorithme général de solution du problème pour obtenir le résultat. Le programmeur n'est pas impliqué dans les étapes qui vont résoudre le problème (l'algorithme).

Les informations fournies à l'ordinateur peuvent être classées en deux catégories: les **faits** et les **règles**. Cette base de connaissance décrit le domaine du problème. Un problème spécifique peut être résolu en posant une question ou **requête**. L'ordinateur examine les requêtes dans le contexte fourni par les règles et les faits et détermine la solution. Par exemple, si les échecs (ou un autre jeu de plateau) représente le domaine de notre problème, les faits peuvent être des choses comme :

- Les différents types de pièces (par exemple pion blanc, pion noir, roi blanc, ...) ;
- Le nombre de pièce de chaque type (8 pions noirs, 1 roi blanc, ...) ;
- La description du plateau et de son organisation (plateau 8x8, 32 cases noires, 32 blanches, …).

Et les règles peuvent par exemple comprendre :

- Les mouvements possibles pour chaque type de pièce (exemple: le fou se déplace selon les diagonales) ;
- Les règles pour déterminer si une pièce est attaquée ou non ;
- Les règles pour déterminer quand une partie est finie et le vainqueur.

Voici le genre de question qu'on peut poser à propos de notre jeu d'échec :

- Étant donné un placement des pièces sur le plateau, quels sont les mouvement disponibles pour une pièce ?
- À partir de la disposition d'un plateau, quelles pièces peuvent bouger ?
- À partir de la disposition d'un plateau, quelles pièces sont attaquées ?

Chaque question précédente correspond à un problème différent mais concret dans le domaine des échecs. Changer le point de vue en passant de "comment résoudre un problème particulier" à "décrire les règles générales d'un problème plus large" nous autorise à chercher des réponses plus variées à des problèmes à l'intérieur du domaine. Dans le reste de cette section, nous illustrerons plus en détail les notions de faits, règles et requêtes en utilisant un exemple de relations familiales. Le but premier est de bien se familiariser avec les mécanismes basiques de la PL.

2.1. Les faits

Les faits sont essentiellement la forme la plus élémentaire d'assertions vérifiées relatives au domaine du problème. Les faits sont aussi connus en tant que données. Prenons une famille de quatre personnes (le fils: Sam, la fille: Denise, le père: Franck, la mère: Mary et le grand-père: Garry) Voici comment on peut décrire les faits afférents à cette famille de façon plus exacte :

1. Sam est un enfant de Mary ;
2. Denise est un enfant de Mary ;
3. Sam est un enfant de Franck ;

4. Denise est un enfant de Franck ;
5. Franck est un enfant de Garry ;

1. Franck est un homme ;
2. Sam est un homme ;
3. Mary est une femme ;
4. Denise est une femme.

Les faits présentés au dessus peuvent être utilisés afin de répondre à des questions simples comme *Est ce que Sam est un homme ?*. Cette question basique dont la réponse est oui ou non peut être résolue en regardant les faits portants sur les genres. Étant donné que nous avons une correspondance exacte avec le fait 2 de la seconde liste, la réponse est *oui* ou encore *vrai*. Cependant, la question *Est ce que Sam est une femme ?* retourne *non* ou *faux*. Ceci s'explique car nous n'avons aucune donnée disant que Sam est une femme. Ainsi la réponse par défaut à une requête est non, a moins qu'une correspondance exacte se fasse.

Un type de question légèrement différent est "Quel est le genre de Sam ?" Cette question n'appelle pas une réponse du type vrai/faux mais on peut y répondre en regardant le fait 2 de la seconde liste. Une autre question pourrait être "Qui est l'enfant de Franck ?". De même, ce n'est pas une réponse vrai/faux, cependant elle est un petit plus intéressante car il n'y a pas une seule réponse mais deux (Sam et Denise). Ceci nous montre qu'on ne peut pas s'arrêter d'examiner les faits dès qu'une réponse est trouvée: on doit continuer les recherches tant que tous les faits intéressants n'ont pas été regardés. Quand il y a de multiples réponses à une question, la personne qui demande peut être intéressée par une réponse, plusieurs ou encore la totalité des réponses.

Une question à laquelle on ne peut pas répondre en regardant juste les faits est "Est-ce que Mary est un parent de Sam ?". On ne peut pas y répondre car nous n'avons pas encore défini ce que signifie être *parent*. Il n'y a aucun fait direct portant sur un quelconque type de relation parent/enfant. Pour résoudre ce problème, nous pouvons ajouter un fait de la nature "X est un parent Y" pour chaque fait du type "Y est un enfant de X" présent au dessus. Cette approche est encombrante et sujette aux erreurs en particulier quand la base de faits est grande. Une meilleur approche est d'utiliser des *règles* pour inférer ces nouveaux faits.

2.2. les règles

Les règles sont des déclarations utilisées pour inférer des nouveaux faits (ou donnés) à partir d'un ensemble de faits existants. Voici quelques règles simples pour décrire les liens de parenté dans la famille:
1) X est un parent de Y si: Y est un enfant de X
2) X est le père de Y si: X est un homme **ET** Y est un enfant de X
3) X est la mère de Y si: X est une femme **ET** X est un parent de Y

Ici, X et Y sont des paramètres et non des constantes comme "Sam" ou "Franck". La règle *parent* fournit la possibilité de répondre à une question du type: "Est-ce que Mary est un parent de Sam ?" ou "Qui est un parent de Sam ?". Il faut noter que les règles *parent* et *père* sont uniquement définies en terme de faits. De l'autre coté, la règle *mère* est spécifiée en utilisant un mélange de faits et de règles bien qu'elle aurait pu être spécifiée d'une manière similaire à la règle *père*

2.2.1. Les règles récursives

Les règles peuvent aussi être spécifiées de manière récursive. Considérons une règle qui définit la notion d'ancêtre:
X est un ancêtre de Y si:
X est un parent de Y **OU**
Z est un parent de Y **ET** X est un ancêtre de Z.

Maintenant, nous pouvons répondre à la question "Est ce que Garry est un ancêtre de Sam ?" qui devrait répondre "vrai". De façon similaire, si nous demandons "Qui est un ancêtre de Sam ?", on devrait obtenir Franck, Mary et Garry.

2.3. Requêtes et assertions

Maintenant que nous disposons de la base de connaissance, nous sommes près à poser des questions. Les problèmes spécifiques à résoudre le sont en posant des questions. Nous avons vu précédemment des requêtes à propos des liens familiaux. D'autres exemples de requêtes peuvent être par exemple lorsque qu'on manipule un graphe "Quel est le chemin le plus court entre deux noeuds A et B ?" ou encore "Est-ce que le graphe G est un graphe cyclique ?"

Les requêtes peuvent se classifier en deux catégories. La première est un simple test pour déterminer si une affirmation est vraie: "Est-ce que Sam est un enfant de Garry ?". Ce sont des assertions étant donné que le travail principal est de vérifier si la sentence est vraie ou non. Le second type de question est de celles qui amènent une ou plusieurs réponses. Par exemple: "Qui est l'enfant de Franck ?", nous ne vérifions pas si un fait est vrai mais nous demandons au système de déterminer les enfants de Franck. Dorénavant, nous parlerons de ces requêtes en terme de requêtes génératives car elles demandent de générer des solutions.

2.4. Calcul par inférence

Jusque là, nous n'avons pas été très précis sur la méthode qui permet de trouver (on utilise aussi inférer) les réponses. Étant donné les faits et les règles décrits avant, il est facile pour n'importe quel individu d'inférer mentalement les réponses aux questions qu'on a posées. Le principe fondamental qui se cache derrière est nommé en logique formelle *modus ponens*

Si A est vrai et que A implique B alors B est vrai (A et B étant des énoncés arbitraires)

Ce principe est le coeur du modèle de calcul utilisé en programmation logique. Une version intuitive mais plutôt rude de l'algorithme utilisé pour répondre à des requêtes dans un système tel que Prolog est le suivant :

- Construire une liste de faits et de règles pertinents ;
- Prendre un fait pertinent et regarder s'il répond à la question. Recommencer tant que la liste des

faits pertinents n'est pas vide ;
- Prendre une règle pertinente qui peut être appliquée pour dériver de nouveaux faits (en utilisant le *modus ponens*). Recommencer jusqu'à tant que la liste des faits et des règles pertinents soit épuisée.

A chacune des étapes de l'algorithme d'inférence, on peut choisir plus d'un fait ou d'une règle pour continuer l'exécution. Chaque choix mène à un chemin d'inférence différent et tous les chemins ne mènent pas forcément à une solution. Les chemins qui ne mènent pas une solution sont abandonnés et le processus d'inférence reprend depuis le point le plus récent où des faits et/ou règles étaient disponibles pour l'inférence. Abandonner le chemin courant et reprendre l'exécution depuis un endroit précédemment exploré afin de faire un choix différent est appelé *backtracking*. Le *backtracking* continue tant que tous les chemins d'inférence n'ont pas été examinés. Ainsi, le calcul est réduit à parcourir des chemins d'inférence déterminés par les faits et les règles disponibles.

Ceci est similaire à effectuer une recherche profonde dans un arbre binaire où les feuilles représentent les résultats finaux possibles et les noeuds internes les résultats intermédiaires. Dans une logique formelle pure, l'ordre de sélection des faits et des règles est non déterminé. Ceci implique que l'ordre dans lequel on va obtenir les réponses est non-déterminé aussi. Étant donné que certains chemins peuvent mener plus rapidement à une solution que d'autres, en pratique l'ordre d'exécution est fixé (le même que l'ordre de déclaration) pour permettre un contrôle sur l'efficacité (i.e. pour améliorer l'efficacité de la recherche). Fixer l'ordre d'application des règles simplifie aussi le raisonnement à propos de l'exécution des programmes logiques ce qui est aussi un point important pour le débogage.

2.5. Résumé

Dans cette section nous avons décrit les faits, règles, requêtes, assertions et l'inférence. Les faits sont simplement des énoncés. Les règles peuvent être utilisées pour dériver de nouveaux faits. Elles peuvent être construites à partir de faits, d'autres règles ou d'elles-mêmes (exemple des règles récursives). Un ensemble de règles et de faits peut être utilisé pour répondre à des questions pertinentes. Les questions peuvent de façon générale être classées en d'un coté celles qui demandent une réponse oui/non et de l'autre celle où on doit générer la réponse. Le premier type de question sont des assertions et le second des requêtes génératives. Les assertions ne peuvent avoir qu'une seule réponse (vrai ou faux). Les requêtes génératives peuvent avoir zéro ou plusieurs solutions. "Est ce que Sam est un homme ?" est une assertion. "Qui est un enfant de Mary ?" est une requête générative. L'inférence logique est utilisée pour répondre aux questions. Elle implique un examen des faits et l'application des règles. De nouveaux faits émergent depuis une application des règles qui deviennent à leur tour candidates pour un examen futur durant le processus d'inférence.

3. Programmation logique en C++

Maintenant, traduisons les faits et règles présentés ci-dessus en C++ de telle sorte qu'on puisse les exécuter. Les exemples sont codés avec Castor, une bibliothèque *Open-Source* qui permet l'utilisation du paradigme logique de façon naturelle en C++ en permettant aux faits et aux règles d'être déclarés en tant que classes, fonctions ou juste expressions. Ce bas niveau d'intégration est très utile et permet un environnement de programmation où les frontières du paradigme perdent leur sens.

Les fonctions C++ suivantes représentent les faits *enfant*, *genre* et la règle *père* (décrite précédemment) en utilisant Castor.

Tout le code relatif à castor se trouve dans le namespace *castor* Pour compiler le code, il faut donc soit faire un *using namespace castor* ou préfixer tous les types avec *castor::*

```
#include "castor.h"
//ou #include <castor.h>, dépend de votre
installation

//c est un enfant de p
relation enfant(lref<std::string> c,
lref<std::string> p)
{
  return eq(c,"Sam") && eq (p,"Mary") //fait 1
   ||  eq(c,"Denise") && eq (p,"Mary") //fait 2
   ||  eq(c,"Sam") && eq (p,"Franck") //fait 3
   ||  eq(c,"Denise") && eq (p,"Franck") //fait 4
   ||  eq(c,"Franck") && eq (p,"Gary") //fait 5
  ;
}

//le genre de p est enregistre dans g
relation genre(lref<std::string> p,
lref<std::string> g)
{
  return eq(p,"Franck") && eq (g,"homme") //fait
1, liste 2
   || eq(p,"Sam") && eq (g,"homme") //fait 2,
liste 2
   || eq(p,"Mary") && eq (g,"femme") //fait 3,
liste 2
   || eq(p,"Denise") && eq (g,"femme") //fait 4,
liste 2
  ;
}
 //f est le pere de c
relation pere(lref<std::string> f,
lref<std::string> c)
{
   //si f est un homme et c un enfant de f
   return genre(f,"homme") && enfant (c,f); //
}
```

Les faits et les règles sont tous deux déclarés en tant que fonctions retournant un objet de type *relation*. Les paramètres sont des instances de la classe générique *lref* qui signifie référence logique (ou *logic reference* en anglais). Ici, la classe fournit un moyen similaire aux références pour passer un objet en paramètre. A l'inverse des références classiques en C++, les références logiques peuvent rester non initialisée. La valeur pointée par une référence logique peut être obtenue en la déférençant avec l'opérateur *.

La fonction *eq* est appelée la relation d'unification. Son

travail est de *tenter* de faire correspondre ses deux arguments. Si l'un de ses deux arguments est une référence logique non initialisée, elle va lui assigner la valeur de l'autre argument. Si les deux arguments ont des valeurs bien définies, alors elle va juste comparer les valeurs. Cette tâche est appelée **unification**. Considérons un appel à *eq(c,"Sam")* dans la relation enfant. Si *c* a été précédemment initialisé, alors *eq* va juste comparer la valeur de *c* avec *"Sam"*. Cependant, si *c* n'a pas été initialisé, on assignera *"Sam"* à *c*. Le type relation, les références logiques et la relation *eq* sont étudiés plus loin dans les section 3.1, 3.2 et 3.3 respectivement.

Aucune des relations (que ce soit *eq* ou une relation définie par l'utilisateur comme *enfant* ou *genre*) ne retourne le calcul voulu au moment où elle est appelées. A la place, elle retourne un objet-fonction qui encapsule le calcul souhaité. L'objet-fonction peut être sauvegardé dans un objet de type relation et l'évaluation du calcul sera déclenché par l'appel de l'opérateur () sur l'objet encapsulant. Avec ces définitions, nous sommes en mesure de faire certaines assertions et certaines requêtes. Le code suivant est une simple assertion pour vérifier si Sam est un homme:

```
relation samIsMale = genre("Sam","homme");
if (samIsMale())
  std::cout<<"Sam est un homme";
else
  std::cout<<"Sam n'est pas un homme";
```

De façon similaire, on peut vérifier si Franck est bien le père de Sam.

```
relation samsDadFranck = pere("Franck","Sam");
if (samsDadFranck())
  std::cout<<"Franck est le père de Sam";
else
  std::cout<<"Franck n'est pas le père de Sam";
```

Nous pouvons aussi faire des requêtes génératives comme "Quel est le genre de Sam ?"

```
lref<std::string> g;
relation samsGender = genre("Sam",g);
if (samsGender())
  std::cout<<"Le genre de Sam est: "<<*g;
else
  std::cout<<"Le genre de Sam est inconnu";
```

Ici nous passons *"Sam"* en tant que premier argument et nous laissons le second indéfini (c'est-à-dire sans lui avoir attribué une valeur). Notez que la même fonction *genre* est utilisée pour vérifier si Sam est un homme et trouver le genre de Sam.

Quand tous les arguments ont été initialisés (c'est à dire qu'ils disposent d'une valeur), le calcul effectue une vérification pour voir s'ils satisfont la relation portant sur le genre. Les arguments qui ont été laissés non initialisés vont agir en tant que paramètres de sortie et être initialisés avec la valeur qui satisfait la relation. Cette nature bidirectionnelle de *lref* en tant que paramètre est essentielle dans le modèle de programmation logique. Il est toutefois important de noter que les références logiques ne sont généralement pas simultanément à la fois des entrées et des sorties de la fonction comme c'est souvent le cas avec le passage par référence pour les fonctions comme std::swap.

Que se passe-t-il si on laisse les deux arguments de la relation genre non initialisés ?

```
lref<std::string> p,g;
relation anyPersonsGender = genre(p,g);
if(anyPersonsGender())
   std::cout<<"Le genre de "<<*p<<" est: "<<*g;
```

Dans ce cas, des valeurs vont être assignées à p et g par la relation *genre*. Étant donné que la première paire déclarée dans la relation est ("Franck","homme"), p sera assigné à *Franck* et g à *homme*.

Les requêtes génératives tel que *samsGender* et *anyPersonsGender* présentées au dessus peuvent avoir zéro, une ou plusieurs solutions. Jusqu'à maintenant, nous n'avons généré qu'une solution, la seule possible. Itérer à travers les solutions utilise assez naturellement l'instruction *while* à la place de l'instruction *if* que nous avons utilisée jusque là. L'exemple précédent peut être ré-écrit de la façon suivante afin de décrire toutes les personnes et leur genre.

```
while(anyPersonsGender())
   std::cout<<"Le genre de "<<*p<<" est:
"<<*g<<std::endl;
```

De façon similaire, on peut lister tous les enfants de Franck:

```
lref<std::string> c;
int count=0;
relation enfantsdeFranck = pere("Franck",c);
while(enfantsdeFranck())
{
  ++count;
  std::cout<<*c<<" est un enfant de
Franck"<<std::endl;
}
   std::cout<<"Franck à "<<count<<" enfants";
```

Une fois que toutes les solutions ont été parcourues, l'appel de *enfantsdeFranck* renvoie *faux* ce qui provoque l'arrêt de la boucle. Quand toutes les solutions ont été parcourues, la référence logique *c* est automatiquement restaurée à son état original de non initialisation.

Il est important de noter comment l'utilisation d'instructions fondamentalement impératives comme *if* ou *while* rendent la transition entre la programmation logique (où les résultats sont générés) et la programmation impérative (où les résultats sont consommés) simple et indolore.

Les fonctions *eq*, le type template *lref* et le type *relation* avec les surcharges des opérateurs && et || fournissent les bases de la programmation logique en C++. Ils sont décrits brièvement dans les sections suivantes. Pour une analyse plus profonde, on peut se reporter au Document: CastorDesign présent sur le site de Castor.

3.1. Le type relation

En programmation logique, il est commun de nommer les faits et les règles en tant que **prédicats** et **relations**. Le terme relation trouve son origine dans la théorie des ensembles où il implique une association entre des ensembles. Ainsi, *genre* est une relation binaire entre un ensemble d'individus et un ensemble de genres. Généralement, une distinction stricte entre règles et faits n'est pas nécessaire lorsqu'on programme avec Castor étant donné qu'ils peuvent être mélangés librement avec la même fonction/expression.

En cohérence avec la programmation logique, nous désignerons dorénavant les fonctions qui représentent les faits ou les règles comme relations. Ainsi, les fonctions retournant un type relation sont aussi appelées relations.

En interne, le type relation représente une fonction ou un foncteur ne prenant aucun argument et qui retourne un booléen. Ainsi, *enfant*, *genre* et *pere* retournent un foncteur qui peut être évalué de manière paresseuse.

3.2. lref: La référence logique

Le type template *lref* est une abréviation pour référence logique (*logic reference* en anglais) et fournit un moyen simple pour faire entrer ou sortir des valeurs d'une relation sous forme d'arguments, de manière similaire aux références du C++. A l'inverse des références du C++, une référence logique ne doit pas forcément être initialisée et fournit la fonction membre *defined* afin de vérifier si elle a été initialisée. L'opérateur de déférencement * et l'opérateur flèche -> peuvent être appliqués à une référence afin d'obtenir la valeur de la référence logique.

Maintenant, regardons la sémantique d'une référence logique lors de son initialisation. Quand une référence logique est initialisée (comprendre construite) ou assignée avec une valeur de type T (ou un type convertible en T), la référence stocke en interne une copie de la valeur. Quand une référence est initialisée avec une autre référence (via le constructeur de copie), on dit que les deux références sont *liées ensembles*. Des références liées ensembles pointent sur la même valeur sous-jacente (s'il y en a une). Ainsi, quand il y le moindre changement à la valeur pointée par l'une des références, ce changement est observé par toutes les références liées ensemble. Une liaison entre des références logiques ne peut être brisée. Cela signifie que si A et B sont des références logiques liées ensembles et que C et D le sont aussi alors on ne peut pas casser la liaison de C vers D pour ensuite la lier à A et B. C continuera à être une part de liaison pour toute sa durée de vie. Les références logiques peuvent seulement être liées par initialisation (construction par copie) mais pas par affectation. Une liaison peut seulement être formée durant la construction de la référence et sera cassée lorsque la référence sera détruite. Quand la dernière référence logique dans une liaison est détruite, la valeur sous-jacente est désallouée.

Ndt: Depuis Castor 1.1, des pointeurs peuvent être utilisés afin d'initialiser une *lref*. Quand on utilise des pointeurs, on doit spécifier si la lref doit gérer ou non la durée de vie de l'objet référencé par le pointeur. Par exemple:

```
//La durée de vie de "Roshan" sera gérée.
lref<std::string> s(new std::string("Roshan"),
true);

//La durée de vie de name ne sera pas gérée.
string name="Naik";
lref<std::string> s2(&name, false);
```

L'affectation de pointeur se fait avec la fonction membre *set_ptr*'

```
std::string str="Castor";
s.set_ptr(&str, false); // désalloue la chaîne
"Roshan" précédemment allouée et ne gère pas la
durée de vie de str
```

Alors qu'utiliser des pointeurs est très utile pour assigner des objets aux lrefs (spécialement quand on mélange plusieurs paradigmes) et efficace, cela peut aussi être très dangereux si on ne les utilise pas avec attention. On doit faire attention à bien spécifier la politique de gestion de durée de vie. Par exemple, si on pointe sur un objet qui est alloué sur la pile, la lref ne doit pas gérer la vie du pointeur, sous peine d'erreur. De façon similaire, si deux lref indépendantes se réfèrent au même objet en utilisant l'initialisation/assignement d'un pointeur, aucune des deux ne doit gérer la durée de vie de l'objet. C'est au programmeur de s'assurer que les objets utilisés par les pointeurs des lrefs sont toujours valides tant qu'elles peuvent tenter d'y accéder. Spécifier accidentellement la politique de gestion de vie à *faux* à la place de *vrai* provoquera une fuite de mémoire.

3.3. La relation eq: La fonction d'unification

La fonction *eq* est la fonction d'unification et prend deux arguments. Les arguments peuvent être des références logiques ou des valeur classiques. *eq* retourne une expression (comprendre un foncteur) qui ,lorsqu'il est évalué, tente d'unifier les deux arguments. Si l'unification réussit, la fonction renvoie *true*, *false* sinon. L'unification effectuée par eq est définie comme suit :

- Si les deux arguments son initialisés, leurs valeurs sont comparées et le résultat de la comparaison retourné ;
- Si seulement un argument est initialisé, l'argument non initialisé se verra modifié et prendra la valeur de l'autre argument ;
- Si aucun des deux arguments n'est initialisé, une exception est lancée.

Ainsi, l'unification va *générer* une valeur pour l'argument non initialisé pour les rendre égaux ou elle va comparer ses arguments s'ils sont tous les deux initialisés. Il est possible d'implémenter d'autres variations de l'unification mais ce n'est en général pas nécessaire.

Les expressions retournées par les divers appels à *eq* à l'intérieur de la relation *genre* par exemple sont rassemblés ensemble pour former une composante d'expression plus grosse à l'aide des opérateurs && et ||. Il est important de noter que le composant de l'expression est retourné sans être évalué. Ces expressions sont évaluées de manière paresseuse. En d'autres mots, ces expressions sont sauvegardées dans un objet de type relation et seront sujettes à une future évaluation quand il sera nécessaire de

la faire. La section suivante examine l'évaluation de ces expressions en détail.

3.4. Évaluation des requêtes

Étant données les relations au dessus, on peut formuler la requête qui vérifie "Est ce que Sam est un homme ?" et enregistrer le résultat dans la variable *samIsMal* comme il suit:

```
relation samIsMale = genre("Sam","homme");
```

L"appel *genre("Sam","homme");* ne réalise en fait aucun vrai calcul, il retourne tout simplement un foncteur qui peut être évalué plus tard afin de vérifier si Sam est un homme. L'appel à une relation ne l'évalue pas. Cette découpe entre appel et évaluation est appelée évaluation paresseuse (*lazy evaluation* en Anglais). Pour exécuter l'évaluation de l'expression, on applique tout simplement l'opérateur () sur la variable qui contient l'expression.

```
if (samIsMale())
  std::cout<<"Le genre de Sam est: "<<*g;
else
  std::cout<<"Le genre de Sam est inconnu";
```

La relation genre a été définie précédemment comme suit:

```
//le genre de p est enregistre dans g
relation genre(lref<std::string> p,
lref<std::string> g)
{
  return eq(p,"Franck") && eq (g,"homme")
   || eq(p,"Sam") && eq (g,"homme")
   || eq(p,"Mary") && eq (g,"femme")
   || eq(p,"Denise") && eq (g,"femme")
  ;
}
```

Après que "Sam" et "homme" ont été passés en tant qu'arguments à *genre*, l'expression retournée ressemble à ce qui suit avec les arguments substitués:

```
   eq("Sam","Franck") && eq ("homme","homme")
   || eq("Sam","Sam") && eq ("homme","homme")
   || eq("Sam","Mary") && eq ("homme","femme")
   || eq("Sam","Denise") && eq ("homme","femme")
```

Cette expression contient quatre expressions && jointes par trois opérateurs ||. Si une seule des expressions && retourne vrai, toute l'expression a été résolue et l'évaluation s'arrête afin de reporter la réussite. Regardons pas à pas l'exécution qui se produit lorsque l'opérateur () est appliqué à *samIsMale*. La première expression *eq("Sam","Franck") && eq ("homme","homme")* est choisie pour être évaluée. On tente d'unifier "Sam" avec "Franck" lorsque on évalue *eq("Sam","Franck")* ce qui échoue étant donné que "Sam" n'est pas égal à "Franck". Les règles de la logique nous permettent de dire que le reste cette expression && n'a pas besoin d'être évalué. Ainsi, ce chemin d'évaluation est immédiatement abandonné le backtracking reprend l'exécution à la prochaine expression && qui est *eq("Sam","Sam") && eq ("homme","homme")* . Cette fois, chaque moitié de l'expression unifie avec succès leurs arguments car ils sont égaux. Une solution a été trouvée et on renvoie vrai à l'appelant qui se trouve être dans l'instruction if.

Si on refait un appel de l'opérateur () sur la variable *samIsMale*, l'exécution reprend où elle s'était arrêtée précédemment. Dans ce cas, il s'agit de la troisième expression && qui échoue sur l'unification de "Sam" et "Mary". Ceci mène à l'évaluation de la quatrième expression && qui échoue aussi pour des raisons similaires et faux sera retourné à l'appelant. Toutes les expressions && ont maintenant été évaluées et appliquer l'opérateur () à *samIsMale* retournera faux immédiatement et ceci de manière immédiate.

Maintenant, considérons comment spécifier une requête générative telle que "Quel est le genre de Sam ?". Une telle question est construite en appelant *genre* avec le premier argument initialisé à "Sam" et en laissant le second non initialisé.

```
lref<std::string> g;
relation samsGender = genre("Sam",g);
if (samGender())
  std::cout<<"Le genre de Sam est: "<<*g;
```

Notons comment il est possible d"utiliser la même relation *genre* pour à la fois vérifier si quelqu'un est un homme ou une femme et trouver le genre d'une personne. Quand une solution sera trouvée, g sera initialisé à "homme". L'expression retournée par *genre("Sam",g)* ressemble à cela après substitution des arguments.

```
eq("Sam","Franck") && eq (g,"homme")
   || eq("Sam","Sam") && eq (g,"homme")
   || eq("Sam","Mary") && eq (g,"femme")
   || eq("Sam","Denise") && eq (g,"femme")
```

Quand l'opérateur est appliqué à samsGender pour la première fois, l'évaluation de la première expression && échoue car on ne peut unifier "Sam" et "Franck". Le processus de backtracking arrive à la seconde expression *eq("Sam","Sam") && eq (g,"homme")* . L'unification de "Sam" avec "Sam" réussit et ensuite *eq(g,"homme")* est évalué. Étant donné que g est non défini, eq va assigner la valeur *"homme"* à g et ainsi l'unification réussit. L'évaluation de l'expression entière s'arrête et renvoie *vrai* au *if*. Si l'opérateur() est à nouveau appliqué sur samsGender, l'exécution reprend au point où elle s'était arrêtée. Cependant une chose très intéressante se produit avant que l'exécution ne reprenne. Il est critique pour la poursuite de l'évaluation que les différents effets de bord qui ont lieu dans la branche abandonnée soient annulés avant de reprendre l'exécution. En effet, si ces effets de bords ne sont pas annulés avant la reprise de l'exécution, ils peuvent affecter les futures évaluations et mener à des résultats faux. Ainsi, l'unification de *g* avec *homme* doit être annulée restaurant ainsi *g* à son état original de non initialisation. Cette fonction de désunification est automatiquement fournie par la fonction d'unification *eq*.

La requête ci-dessus n'a qu'une seule solution mais nous avons observé précédemment que des requêtes peuvent avoir plusieurs solutions. Par exemple, on peut vouloir trouver tous les hommes du système. Une fois de plus, on va utiliser la relation *genre* mais on va laisser le premier argument non initialisé et le second argument à *homme*.

```
lref<std::string> person;
relation males = genre(person,"homme");
while(males())
   std::cout<<*person<< " est un
homme"<<std::endl;
```

Cette fois, on appelle *males()* jusqu'à ce qu'elle retourne *faux*. Chaque appel à *males* déclenche la recherche de la prochaine solution, c'est-à-dire une valeur acceptable pour la référence *person*. A chaque tour de boucle, on assigne à *person* une valeur différente qui représente une solution. Quand toutes les solutions ont été trouvées, *males* retourne faux, la boucle s"arrête et le processus de backtracking restaure *person* à son état original de non initialisation. Etant donné que *person* ne se refère plus à rien après que la boucle soit finie, tenter d'accéder à sa valeur pas le biais de l'opérateur * ou de l'opérateur -> lancera une exception.

3.5. Les règles récursives

La récursion est souvent quelque chose d'essentiel quand on déclare des règles dans le paradigme logique. Dans notre exemple des relations familiales, considérons la définition d'une règle portant sur la relation "ancêtre". Notez qu'il peut y avoir un nombre quelconque de niveaux parent-enfant entre l'ancêtre et le descendant. On peut définir cette règle de façon récursive:

A est un ancêtre de D si:
D est un enfant de A OU
D est un enfant de P ET A un ancêtre de P.

Le code suivant est la traduction naïve de la règle ci-dessus.

```
//Version défectueuse
relation ancetre(lref<std::string> A,
lref<std::string> D)
{
   lref<std::string> P;
   return enfant(D,A) || enfant(D,P) && ancetre
(A,P);
}
```

Cependant, cette définition contient une récursion infinie. L'instruction return contient un appel récursif à ancêtre. Afin de briser la récursion, on a besoin de repousser l'appel récursif de telle sorte qu'il n'ait lieu que seulement lorsqu'on en a vraiment besoin. Castor fournit une relation aidant à cela avec *recurse* qui sert à définir des règles récursives. *recurse* prend en premier paramètre la relation qui doit subir un appel récursif et ensuite les paramètres de cette relation qui sont nécessaires à son appel. La relation retourne un foncteur qui lorsqu'il est évalué mène à l'actuel appel récursif d"*ancetre*. Le problématique appel *ancetre(A,P)* est tout simplement ré-écrit en *recurse(ancetre,A,P)*:

```
relation ancetre(lref<std::string> A,
lref<std::string> D)
{
   lref<std::string> P;
   return enfant(D,A) || enfant(D,P) &&
recurse(ancetre,A,P);
}
```

Dans l'exemple au dessus, la récursion a lieu sur la relation *ancetre* qui est définie en tant que fonction libre. L'usage de *recurse* pour des fonctions membres statiques est exactement le même. Pour utiliser *recurse* sur des fonctions membres, il faut passer en plus le pointeur *this* comme il suit:

```
recurse(this,&Type::fonction, ...arguments...);
```

Retrouvez la suite de l'article de Roshan Naik traduit par David Come en ligne : Lien46

Les dernières news

C++0x : Le Draft final a été voté!

Herb Sutter nous fait part sur son blog (Lien47) de la conclusion du dernier vote du comité qui s'est déroulé à Pittsburgh.

La principale bonne nouvelle est que le Final Commitee Draft a été voté et ne sera donc changé que pour des corrections de bugs et autres typos. Autrement dit, il ne reste qu'à faire valider le draft par ISO et nous auront enfin la nouvelle norme fixée.

D'après Herb, le temps que cela se passe, nous serons en 2011.

Parmi les derniers changements, la suppression définitive d'export template mais surtout la déprécation des spécifications d'exception et l'ajout d'un qualificateur : *noexcept*.
A priori ça sera bien plus intéressant que les spécification d'exceptions.

Commentez cette news en ligne : Lien48

Que penser des frameworks et architectures qui font dériver toutes les classes d'un SuperObjet ?

Salut,
Beaucoup de framework, notamment IHM (Qt, wxWidgets, MFC) sont construits autour d'une super classe de laquelle héritent toutes ou parties des classes du framework. Que pensez-vous de cette pratique ? Est-ce une bonne chose dans l'architecture de vos logiciels ?
Personnellement, je pense qu'il s'agit d'une pratique présentant plus d'inconvénients que d'avantages. Et vous ?

Commentez cette news en ligne : Lien49

Les dernières news

Qt 4.7.0 Tech Preview et Qt Creator 2.0 alpha

Sortie de Qt dans sa version 4.7.0 Tech Preview et de Qt Creator 2.0 alpha

Qt 4.7.0 Tech Preview

Moins d'un mois après la sortie de la version 4.6.2 du framework Qt, Nokia nous propose son framework en version 4.7.0 Tech Preview.

Pour rappel, une Tech Preview est la première étape dans le processus de sortie d'une nouvelle version. L'objectif est de présenter les nouvelles fonctionnalités aux utilisateurs en attendant un maximum de retours d'expérience pour améliorer la stabilité et corriger d'éventuels bugs. Il ne s'agit donc pas d'une version à utiliser en production.

La sortie de la version finale 4.7.0 est prévue pour la mi-2010.

Les éléments les plus attendus dans cette version sont :
- le développement d'interfaces graphiques déclaratives avec Qt Quick ;
- la nouvelle API multimédia ;
- des nouveautés dans le module Qt Network ;
- de nouvelles classes, fonctions, macros...

La liste complète des nouveautés (Lien50).

Qt Creator 2.0 alpha

Un an après la sortie de Qt Creator 1.0, voici Qt Creator 2.0 dans sa version alpha annoncée hier par Nokia.

Il s'agit cette fois d'une version 2.0 et non 1.4 du fait de l'ajout de 2 fonctionnalités majeures :
- L'intégration de Qt Quick ;
- Le développement sur Symbian et Maemo

Une petite vidéo (Lien51) vaut mieux qu'un long discours.

Voir aussi

Le site web de Qt (Lien52)
Qt Quick (Lien53)
Sortie de Qt 4.6 (Lien54)

Et vous ?

Allez-vous tester cette nouvelle version de Qt ? Que pensez-vous du développement sur Symbian et Maemo avec Qt Creator ? Êtes-vous prêt à passer aux interfaces graphiques déclaratives avec Qt Quick ?

Commentez cette news en ligne : Lien55

Les derniers tutoriels et articles

De QObject aux méta-objets, une plongée au cœur des fondations de Qt

Qt est un framework dont les fondations ne sont pas connues de la totalité des gens qui l'utilisent. Bien entendu, il est possible de se demander en quoi un codeur d'interfaces graphiques peut avoir besoin d'utiliser explicitement les méta-objets dans son code. Il est toujours intéressant de savoir sur quoi est fondé un framework. Quel est le modèle objet utilisé ? Que sont les méta-objets ? Ce sont des questions auxquelles cet article a pour but de répondre.

1. Introduction

Comme vous devez l'avoir compris en vous lançant dans la lecture de cet article, j'ai pour objectif de vous en apprendre plus sur les fondations de Qt, les piliers mêmes de ce framework (ensemble de bibliothèques) aux allures si prometteuses. Pour arriver à mes fins, je vous ferai étudier une grande partie de ce qui existe, depuis la classe QObject jusqu'aux méta-objets, en passant bien sûr par les signaux, les slots et bien d'autres choses encore.

Comme cet article a pour but d'être informatif, nous verrons uniquement des extraits de code permettant d'illustrer ce qui y est dit, donc de servir de support aux explications. Ainsi, nous n'étudierons pas ou uniquement à titre exceptionnel des « gros » codes.

Je précise que, même si je vous transmettrai ici de nombreuses connaissances, je ne pourrai me montrer parfaitement exhaustif. Toutefois, si vous relevez dans cet écrit une erreur, un oubli quelconque méritant d'être corrigé ou autre, je vous prie de bien vouloir me contacter par message privé (Lien56), je serai ravi de pouvoir enrichir son contenu.

2. À propos de QObject

2.1. Pourquoi QObject est-il à la base de Qt ?

Comme vous le savez déjà, Qt est orienté objet et qui dit POO (Programmation Orientée Objet), dit forcément héritage. Le principe de l'héritage est plus ou moins le même que celui d'un arbre généalogique comportant des classes, qui héritent elles-mêmes d'autres classes : nous en arrivons forcément à une classe de base située à la racine

de cet arbre, que l'on appelle aussi classe mère ou classe parente. Dans le cas de Qt, il s'agit de QObject.

Cette classe a pour avantage de ne correspondre à rien de « graphique ». Cela permet à Qt de gagner en puissance car elle fournit au final nombre d'utilitaires, comme le système de méta-objets dont nous allons longuement parler dans cet article sans pour autant être exhaustif, comme dit dans l'introduction. Ces utilitaires sont pour la plupart très ciblés sur le concept d'objet. QObject a donc indéniablement plus sa place que les autres classes actuellement existantes comme pilier central du framework Qt.

QObject, même si c'est la classe mère, n'est pas la classe parente de toutes les autres : certaines d'entre elles, comme QString, n'ont pas besoin des outils qu'elle propose.

2.2. Qu'en est-il du modèle objet de Qt ?

À la différence du C++, de nature plutôt statique, Qt fournit une efficacité d'exécution nécessaire à la réalisation d'une interface utilisateur, à un niveau de flexibilité relativement élevé.

Le modèle objet de Qt, axé sur QObject et sur le moc, qui sera détaillé plus tard dans l'article, offre de nombreux avantages et utilités dont les principaux sont les suivants :

- une gestion de la communication entre les objets par le biais d'un système le plus souvent appelé mécanisme de signaux et de slots (partie IV) ;
- une hiérarchie logique aux utilités multiples entre les objets, telles que les conversions de type (partie III) ;
- un système événementiel relativement utile lors du développement d'interfaces utilisateur (Qt est parfois considéré comme événementiel) ;
- une gestion de la mémoire aisée pour les développeurs, notamment par une destruction de la totalité des objets enfants lorsque l'objet parent est détruit ;
- l'introspection et les propriétés (partie V).

La documentation insiste sur le fait que les objets de Qt doivent être considérés comme des éléments possédant une identité, non comme des valeurs. Malgré les apparences, il faut noter que le modèle objet de Qt se base comme n'importe quelle application codée en C et/ou en C++ sur le C++, ce qui explique en partie la portabilité du framework.

3. Qu'est-ce que Q_OBJECT ?

Q_OBJECT est une macro, un pseudo-programme permettant par le biais d'un identifieur de remplacer des morceaux de codes tels que des defines par des valeurs, fonctions ou autres. Cette macro permet à la classe la contenant dans sa section privée de posséder un méta-objet, à condition qu'elle hérite directement ou non de QObject (pour le cas où la classe ne serait pas dérivée de QObject, il existe aussi la macro Q_GADGET, toutefois très peu documentée).

Le fait de l'employer à l'intérieur d'une classe permet l'utilisation de ce que fournit le système de méta-objets, comme par exemple les signaux et les slots qui ne seraient pas disponibles sans cela. Si vous souhaitez en apprendre plus sur ce que fournit la macro, consultez la partie V qui traite le sujet des méta-objets.

Il faut noter que certaines méthodes comme tr(), inherits() et autres nécessitent la présence de la macro pour fonctionner correctement, et c'est pour cela qu'il est recommandé à plusieurs reprises par la documentation de l'utiliser « dans toutes les sous-classes de QObject sans prêter attention au fait qu'elles utilisent actuellement ou non les signaux, les slots et les propriétés ».

4. Une question d'héritage

Veuillez noter avant tout que chaque partie de cet article est unie par les méta-objets. Cette partie de l'article est donc en continuité avec le reste de celui-ci.

4.1. Conversions de type et QObject

À titre de rappel, une conversion de type avec Qt retourne, en cas de réussite, un objet de type particulier d'un objet subissant l'opération. Par rapport aux conversions de type du C++, celles de Qt sont légèrement différentes car elles ne nécessitent pas le support RTTI (Run Time Type Information) (Lien57).

Dans cette sous-partie, nous allons considérer deux exemples de conversions de type assez similaires :

- qobject_cast ;
- qvariant_cast.

Quel est le but de ces deux présentations ? Tout simplement de vous montrer l'utilité de Q_OBJECT pour pouvoir aborder le concept de méta-objet sans pour autant que vous vous heurtiez à l'inconnu.

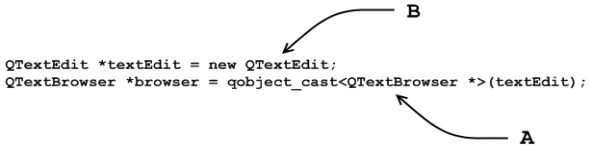
4.1.1. La conversion de type qobject_cast

```
A objet = qobject_cast<A>(B) ;
```

Cette conversion de type est un cast de pointeur qui fonctionne à travers les branches de l'héritage : elle retourne un objet de type A à partir d'un objet de type B, mais sous certaines conditions. Tout d'abord, le type A doit être un type dérivé du type B. Par exemple, le code suivant n'est pas correct :

```
QTextEdit *textEdit = new QTextEdit;
QTextBrowser *browser = qobject_cast<QTextBrowser *>(textEdit);
```

Dans ce cas, la conversion de type retournera 0 car la condition énoncée ci-dessus n'est pas respectée : QTextEdit n'est pas une classe dérivée de QTextBrowser, c'est l'inverse, donc une incompatibilité est présente.



Toutefois, une conversion de type est réversible, elle permet de repasser d'un type à l'autre, à condition que la

conversion précédente n'ait pas retourné 0. La seule nécessité est donc qu'il y ait un rapport d'héritage entre les deux types.

```
QTextEdit *textEdit = new QTextEdit;
QObject *casted_textEdit = qobject_cast<QObject *>(textEdit);
QTextEdit *recasted_textEdit = qobject_cast<QTextEdit *>(casted_textEdit);
```

Ce qu'il faut tout de même savoir est que qobject_cast ne fait pas de distinction entre les types personnalisés et les types inclus par défaut avec Qt (exemple : QTextEdit).

Il peut être intéressant de présenter ici la fonction QObject::inherits() qui retourne un booléen dont la valeur dépend de l'héritage d'une classe ou non.

```
QTextEdit *textEdit = new QTextEdit;
bool a = textEdit->inherits("QWidget"); // a == true
bool b = textEdit->inherits("QTextBrowser"); // b == false
```

La seconde condition est que la classe A soit déclarée avec la macro Q_OBJECT. Dans le cas inverse, la valeur de retour serait indéfinie, si le compilateur ne signalait pas une erreur. En effet, qobject_cast requiert les méta-informations fournies par le système de méta-objets pour fonctionner correctement.

D'un point de vue pratique, qobject_cast est très utile lors de l'utilisation de sender(). Placé dans un slot, cette fonction renvoie un QObject de l'objet ayant envoyé le signal l'appelant et la conversion de type qobject_cast permet d'en récupérer un widget du type d'un QPushButton. Nous verrons plus amplement cela dans la partie sur les signaux et les slots.

4.1.2. La conversion de type qvariant_cast

QVariant est une classe permettant de faire le lien entre la plupart des types de données, tels que uint, bool, QString, etc. Il permet par le biais de fonctions telles que toString() d'exprimer sa valeur sous différents types de données, mais dès qu'il s'agit de passer de QVariant à QColor, par exemple, et bien il faut nécessairement passer par une conversion de type, qui est ici qvariant_cast. On aurait bien évidemment pu passer par autre chose mais je préfère ne parler que de cela ici.

D'un point de vue syntaxique, cette conversion de type est identique à qobject_cast, mais il s'agit bien de l'un des rares points communs entre les deux conversions de type. En effet, elle ne prend pas en compte l'héritage car il est impossible de parler de cela pour des types comme int et bool. Comme l'héritage n'a aucun rôle ici, le système de méta-objets n'en a donc pas non plus. Ainsi, la macro Q_OBJECT n'influe en rien sur qvariant_cast. Au final, on ne parle que de compatibilité entre les types. Si cette compatibilité n'est pas présente, la valeur de retour de la conversion sera, comme pour qobject_cast, 0.

```
QVariant color = QColor(0, 0, 0, 0);
qDebug() << qvariant_cast<QColor>(color); // QColor(ARGB 0, 0, 0, 0)
qDebug() << qvariant_cast<QString>(color); // #000000
qDebug() << qvariant_cast<quint32>(color); // 0
```

On peut donc déduire de la présentation des deux conversions de type que même si le système de méta-objets est relativement présent, Qt ne repose pas totalement dessus.

5. Allons plus loin avec les signaux et les slots

5.1. Le principe

Depuis sa création, Qt propose un mécanisme ingénieux couramment appelé système de signaux et de slots. Ce système est relativement pratique quand il est présent dans des applications graphiques car il permet la communication entre plusieurs objets. Comme la documentation recommande de parler d'eux comme des identités, non comme des valeurs, nous pouvons considérer l'exemple d'un bouton sur lequel l'utilisateur cliquerait. Si l'intérêt de la présence de ce bouton était de quitter l'application lors d'un clic dessus, l'interaction entre ce bouton et l'objet de l'application, qApp, serait d'appeler une fonction de fermeture, quit(). Dans ce cas de figure, l'objet *émetteur* serait le bouton et l'objet *receveur* serait qApp. Pour créer une interaction entre ces deux objets, il faudrait créer une connexion entre eux. Par chance, Qt fournit la méthode statique et thread-safe connect() qui permet de mettre en place ce type de connexion.

```
connect(émetteur, signal, receveur, slot) ;
```

Comme vous pouvez le constater, connect() prend quatre arguments dont nous connaissons l'émetteur et le receveur, mais il accepte aussi un cinquième, le type de connexion, dont nous ne parlerons pas dans cette partie. L'argument signal correspondrait dans l'exemple du bouton à un clic, donc au signal clicked() de QPushButton. Un signal peut être considéré comme une condition à l'appel de son slot. Quant à lui, le slot correspondrait à l'action produite lors d'une interaction entre l'émetteur et le receveur, soit ici à la fermeture de l'application, donc à l'appel de la fonction quit(). Plus simplement dit, c'est une fonction appelée lors de l'interaction.

```
QPushButton *bouton = new QPushButton("Quitter");
connect(bouton, SIGNAL(clicked()), qApp, SLOT(quit())) ;
```

Une connexion entre deux objets requiert le système de méta-objets, donc la présence de la macro Q_OBJECT dans la section privée de la classe où est faite la connexion. Toutefois, le slot quit() est un slot implémenté par défaut avec Qt et non un slot créé par l'utilisateur. Le système de méta-objets n'est donc pas nécessaire pour ce cas de figure. Sans le système de méta-objets, connecter deux objets par le biais d'un signal personnalisé et/ou un slot personnalisé n'aura aucun effet. Notez que le nombre de connexions d'un objet avec un autre n'est pas limité.

De la même manière que l'on peut connecter entre eux deux objets, il est possible de les déconnecter. Cette fois-ci, c'est la fonction disconnect() qui entre en jeu, et avec les mêmes arguments que ceux entrés dans la fonction connect().

```
disconnect(bouton, SIGNAL(clicked()), qApp,
SLOT(quit())) ;
```

Comme vous pouvez le constater dans les codes présentés, il n'est pas nécessaire que l'émetteur reçoive des informations de la part du receveur et vice versa : les deux objets restent indépendants l'un de l'autre lors d'une connexion. Il est d'ailleurs possible d'émettre une communication de valeurs lors d'une connexion/déconnexion. Un exemple qui illustrerait assez bien le principe serait une connexion entre un slider et une barre de progression : quand la valeur du slider changera, la barre de progression prendra la même valeur que celle du slider :

```
QSlider *slider = new QSlider(Qt::Horizontal);
QProgressBar *progressBar = new QProgressBar;
connect(slider, SIGNAL(valueChanged(int)),
progressBar, SLOT(setValue(int)));
```

Il faut garder en tête que le mécanisme de signaux et de slots est extrêmement flexible. Selon la documentation, il est possible d'émettre environ deux millions de signaux par seconde, quand ils sont connectés à un même receveur, sous un i586-500 (la 5° génération de Pentium).

5.2. Signaux et slots personnalisés

Jusqu'alors, nous n'avons vu que des signaux et des slots présents par défaut avec Qt. Ces signaux, même s'ils sont utiles, ne sont pas suffisants pour combler tous les besoins des programmeurs. C'est pour cela qu'il est possible de créer et de gérer ses propres signaux et slots. Comme dit dans la sous-partie précédente, l'utilisation de signaux et/ou de slots personnalisés nécessite la présence de la macro Q_OBJECT dans la section privée de la classe où ils sont gérés.

5.2.1. Les signaux personnalisés

La création de signaux se fait lors de la déclaration de la classe. On déclare un signal comme on déclare une fonction, soit par un simple prototype. La différence entre les deux est que le programmeur ne gère pas l'implémentation d'un signal, c'est le système de méta-objets qui s'en occupe. En C++, une fonction se déclare dans le header sous le mot-clé public ou private. Avec Qt, un signal se déclare sous un autre mot-clé : signals.

```
class Label : public QLabel
{
        Q_OBJECT
        signals:
        void released();
        // Reste de la déclaration
};
```

J'ai choisi de vous présenter un exemple de classe dérivée de QLabel pour vous montrer comment créer un signal released(), déclenché lorsque le bouton de la souris est relâché après un clic sur celui-ci. J'aurais pu choisir de vous présenter un exemple avec un signal nommé pressed() mais dans la majorité des cas, on préfèrera utiliser le relâchement au lieu de l'enfoncement. Pour cela, la seule chose à faire est d'appeler le signal released() par le biais d'une instruction particulière, emit, lors de l'appel de l'évènement mouseReleaseEvent().

```
void Label::mouseReleaseEvent(QMouseEvent *event)
{
        Q_UNUSED(event);
        emit released();
}
```

De là, il est possible d'effectuer une connexion des plus basiques :

```
connect(this, SIGNAL(released()), qApp,
SLOT(quit()));
```

Bien entendu, il est possible de transmettre des valeurs lors de l'émission du signal personnalisé. Par exemple, dans le cas présenté, passer en argument la position de la souris lors de l'évènement est possible.

```
emit released(event->pos());
```

Le prototype devra, en conséquence, contenir un argument de type QPoint pour que la compilation se déroule sans problème. Ce qu'il faut tout de même retenir est qu'il est impossible de passer une valeur lors de la connexion, la valeur est uniquement transmise lors de l'émission du signal. Ce code est donc incorrect :

```
connect(this, SIGNAL(released(QPoint(12, 12))),
qApp, SLOT(quit())); // Incorrect
```

Un équivalent correct de cette connexion pourrait uniquement être mis en place lors de l'appel de l'instruction emit, non ailleurs.

De même, il est incorrect de placer un nom de variable dans une connexion : il faut uniquement placer le type.

```
connect(this, SIGNAL(released(QPoint point)),
qApp, SLOT(quit())); // Incorrect
```

Ainsi, la création de signaux est extrêmement simple : une déclaration et l'utilisation de l'instruction emit. Le système de méta-objets permet dans le cas présent au programmeur de ne pas avoir à implémenter le signal. Toutefois, si vous vous demandez où est placée cette fameuse implémentation, regardez le fichier moc_[Nom du fichier d'en-tête].cpp généré par le moc (appelé explicitement : projet Visual Studio et autres, ou implicitement : qmake, etc.). Elle y est faite sous le même prototype que celui que vous avez déclaré dans votre header pour le signal dont il est question.

Maintenant que je vous ai parlé des signaux personnalisés, je peux vous parler d'un autre aspect de la fonction connect(), bien que nous n'en verrons pas la totalité : il est possible de connecter deux signaux entre eux. Une telle connexion ferait que lorsqu'un signal est émis, l'autre l'est aussi. Pour vérifier la véracité de mes dires, testez ces deux lignes (deux signaux y sont utilisés : released() et released(QPoint)) :

```
connect(this, SIGNAL(released(QPoint)), this,
SIGNAL(released()));
connect(this, SIGNAL(released()), qApp,
SLOT(quit()));
```

Lorsque le signal released(QPoint) est émis, released() l'est aussi, ce qui engendre la fermeture de l'application.

Une dernière chose sur les signaux : il est possible qu'à un moment ou à un autre vous souhaitiez, pour une raison ou une autre, qu'un objet cesse d'envoyer des signaux. Pour cela, il suffit d'appeler la méthode blockSignals() en passant en argument false. Dès lors, l'objet n'enverra plus aucun signal.

5.2.2. Et les slots personnalisés

Les slots ne sont rien de plus que des fonctions standards ayant la capacité d'être appelées par des signaux lors de connexions : ils peuvent sans problème être appelés normalement, être virtuels ou autres. Par conséquent, la création d'un slot se fait exactement comme la déclaration d'une fonction, donc avec un prototype et une implémentation, sauf que leur déclaration se situe sous l'institution public slots, private slots ou encore protected slots, afin de les différencier des fonctions normales.

```
public slots:
void about();
```

Dans le slot about(), présent en tant que slot public, il est par exemple possible d'ouvrir une boîte de dialogue, les possibilités ne manquent pas :

```
void MainWindow::about()
{
        QMessageBox::information(this, "Boîte de dialogue", "Cette boîte de dialogue "
                                 "correspond à une implémentation basique d'un slot personnalisé.");
}
```

Plus tôt, nous avons vu que les connexions entre signaux étaient possibles. Ce n'est pas le cas pour les slots car une connexion nécessite au minimum un signal. D'ailleurs, les slots ont eux aussi besoin du système de méta-objets pour fonctionner correctement : la présence de la macro Q_OBJECT est donc une fois de plus nécessaire.

Dans l'univers de Qt, il existe ce qu'on appelle communément des connexions automatiques. Ce type de connexions est, par exemple, présent dans les IU (Interface utilisateur) avec les slots dont les prototypes sont architecturés sous la forme suivante :

```
void on_[Nom de l'objet]_[Nom du signal]
([Arguments]);
```

Nous ne verrons pas en détail ce type de connexions.

La méthode sender() a été introduite dans la partie sur la conversion de type qobject_cast. Comme elle fonctionne dans le cadre d'un slot, il peut être intéressant de la présenter dès maintenant. Cette méthode renvoie un QObject de l'objet ayant envoyé le signal appelant le slot dont il est question. Cette particularité permet d'effectuer de multiples connexions sur le même slot, ce qui peut être très pratique dans certains cas où les connexions seraient répétitives. Voici un court exemple permettant de vérifier si le texte de l'émetteur d'un slot, présumé être un QLineEdit, correspond à un chemin de fichier existant :

```
void MainWindow::verif()
{
        QLineEdit *lineEdit = qobject_cast<QLineEdit *>(sender());
        if(lineEdit && QFileInfo(lineEdit->text()).exists())
        QMessageBox::information(this, "Vérification", "Le chemin est valide.");
}
```

5.3. QSignalMapper

Il arrive parfois qu'un développeur utilisant Qt ait besoin d'effectuer de multiples connexions sur le même slot. L'alternative à effectuer des connexions fastidieuses vers des slots ayant le même effet ou bien à passer par la méthode sender() serait de passer par QSignalMapper, qui permet assez facilement de réaliser ce type de connexions, en renvoyant en même temps un identifiant de l'émetteur, tel un QString, un int, un QWidget ou encore un QObject. Retourner un QWidget ou un QObject peut être très pratique une fois combiné avec une conversion de type telle que qobject_cast.

Pour réaliser des connexions multiples sur le même slot avec QSignalMapper, la connexion de deux signaux peut être très intéressante à mettre en œuvre. Je pense que l'heure est venue de vous montrer un court exemple d'utilisation :

```
MainWindow::MainWindow()
{
        setCentralWidget(new QWidget);
        QVBoxLayout *layout = new QVBoxLayout;
        centralWidget()->setLayout(layout);

        QSignalMapper *mapper = new QSignalMapper(this);
        connect(mapper, SIGNAL(mapped(int)), this, SIGNAL(textChanged(int)));
        connect(this, SIGNAL(textChanged(int)), this, SLOT(verif(int)));

        for(int i = 0; i < 10; i++)
        {
                lineEdit.append(new QLineEdit(QString::number(i + 1)));
                mapper->setMapping(lineEdit[i], i);
                connect(lineEdit[i], SIGNAL(textChanged(QString)), mapper, SLOT(map()));
                layout->addWidget(lineEdit[i]);
        }
}

void MainWindow::verif(int index)
{
        QString text = lineEdit[index]->text();
        if(QFileInfo(text).exists())
        QMessageBox::information(this, "Vérification", "Le chemin du champ "
                                 + QString::number(index + 1) + " est valide.");
}
```

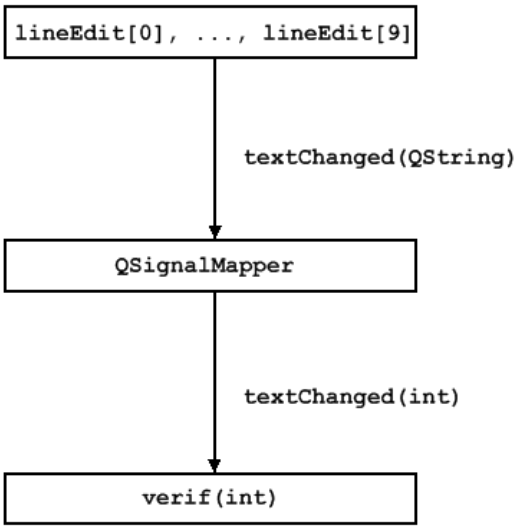
Ce code fonctionne avec trois éléments préalablement déclarés dans l'en-tête :

- le signal textChanged(int) ;

- le slot verif(int) ;
- la liste QList <QLineEdit *> lineEdit.

Dans mon code, je déclare un QSignalMapper qui permettra les multiples connexions grâce à sa capacité de constituer une interface entre plusieurs signaux et un slot possédant un seul paramètre. Grâce à la connexion présente dans la boucle for, lorsque le texte d'un des QLineEdit de la liste change, le slot map() est appelé. Le fait d'utiliser la fonction setMapping() avec comme second argument un int permet de faire en sorte que le signal mapped(int) soit émis lorsque map() est appelé. Pour finir, je connecte ensuite ce signal au slot verif(int).

Si on regarde la chose globalement, les choses se font comme sur le schéma suivant :



Dès lors, nous pouvons passer à la partie sans doute la plus importante de l'article, celle sur le système de méta-objets.

6. Au cœur des méta-objets

6.1. L'intérêt du système de méta-objets

Une grande question peut se poser : pourquoi utiliser un tel système ? En réalité, la principale raison de sa présence est le besoin d'instaurer un mode de communication entre les objets, qu'ils aient ou non un lien entre eux (nous verrons d'ailleurs, dans la partie suivante, que les conséquences de la présence d'un tel système sont les critiques dont il est l'objet). D'après ce que nous avons vu et ce que nous allons voir, le système de méta-objets est nécessaire aux dispositifs suivants :

- le système de signaux et de slots ;
- les informations de type disponibles lors de l'exécution, donc l'introspection ;
- le système de propriétés.

6.2. Méta-objets et templates

Le fait que Qt utilise un outil additionnel, le moc (traité à la suite) et qu'il utilise une implémentation basée sur des macros fait l'objet de critiques, car il rend toute programmation avec Qt différente d'une programmation pure C++. Il s'avère que les développeurs de Qt voyaient cette utilisation comme une nécessité pour pouvoir fournir le mécanisme de signaux et de slots ainsi que les dispositifs introspectifs, non comme un moyen de se démarquer du C++.

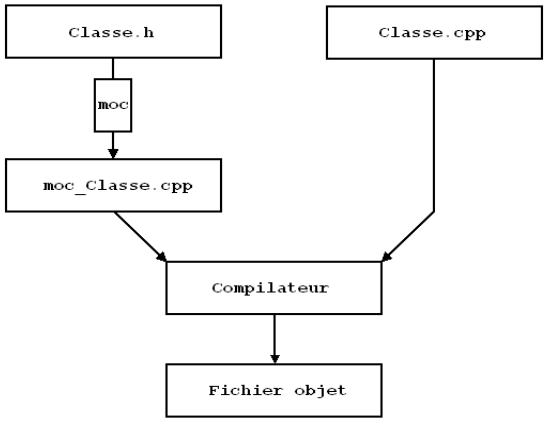
Toutefois, certaines personnes pensent que les templates, déjà largement utilisés dans Qt, seraient plus appropriés pour gérer cela. Dans cette page (Lien58), la documentation explique en détail pourquoi l'utilisation d'outils additionnels est préférable à l'utilisation des templates.

Vous pouvez aussi retrouver un débat complet sur la question (Lien59). Ce débat a pour base une opposition de GTK et de Qt et s'oriente progressivement vers le sujet des méta-objets, en passant et en repassant par le sujet des signaux et des slots. La principale critique émise est notamment le fait énoncé plus haut, précisant que toute programmation avec Qt est différente d'une programmation pure C++ (fait contesté : il est possible d'éviter de diverger d'une programmation pure C++). L'intérêt du moc est remis en cause à plusieurs reprises.

6.3. Qu'est-ce que le moc ?

Le moc est le compilateur de méta-objets. Make se sert du moc lorsqu'il est appelé, mais celui-ci peut aussi être appelé manuellement. Son fonctionnement est simple : il lit les fichiers sources et retrouve les déclarations de classe où la macro Q_OBJECT est utilisée. Une fois cela terminé, le moc va générer des fichiers reconnaissables par leur nom sous la forme de moc_[Nom du fichier d'en-tête].cpp (dans le cas d'une utilisation de Q_OBJECT dans un fichier d'en-tête) ou bien sous la forme de [Nom du fichier].moc (dans le cas d'une utilisation de la macro sans fichier d'en-tête (Lien60)), qui contiendront le code de méta-objets, par exemple nécessaire à l'utilisation des signaux et des slots.

Voici un schéma illustrant les étapes de la compilation de deux fichiers respectivement nommés Classe.cpp et Classe.h avec un compilateur quelconque. Ce dernier fichier contient dans le cas présent, à la différence du fichier Classe.cpp, une déclaration de classe dans laquelle la macro Q_OBJECT est employée.



Bien entendu, le fichier objet de moc_Classe.cpp est lui aussi généré après l'utilisation du compilateur car ce fichier est bel et bien considéré comme un fichier source.

Ce système a beau être attrayant, il possède tout de même des points faibles. Par exemple, il ne gère pas correctement les classes templates qui se voient donc privées des signaux et des slots. La documentation détaille ici les limitations que cause le moc (Lien61).

6.4. Les propriétés

Les propriétés constituent un dispositif multiplateforme s'avérant être le pilier de la majorité des classes de Qt utilisant le système de méta-objets. Elles prennent d'ailleurs base sur ce système. Afin d'expliquer clairement comment s'en servir, nous allons étudier un petit code contenant une classe nommée MyWidget, héritant de QWidget et ayant la macro Q_OBJECT définie dans sa section privée. Ci-dessous se trouve une première utilisation des propriétés avec la propriété « windowTitle » :

```
MyWidget widget;
widget.setProperty("windowTitle", "MyWidget");
```

Ici, nous instancions notre classe MyWidget et nous définissons la propriété windowTitle avec la valeur MyWidget. Une alternative plus commune de l'utilisation de setProperty consiste à utiliser la fonction setWindowTitle. Toutefois, utiliser cette fonction à la place de setProperty ne contourne pas l'utilisation du système de propriétés, car elle s'avère être ce qu'on appelle la fonction d'écriture WRITE de la propriété windowTitle, dont nous allons parler à la suite.

Pour paraître plus concret, les propriétés constituent un outil utilisé par les modules QtScript et QtDBus pour déterminer ce qui peut être exporté depuis l'objet au script/à l'environnement D-Bus, ainsi que par le module QtDeclarative pour déterminer ce qui sera exporté depuis une classe dans QML.

Cette courte introduction a pour objectif de vous familiariser avec les propriétés. Nous allons voir en détail comment définir une propriété (et cela en premier lieu car nécessaire à la compréhension des propriétés fournies par défaut), comment gérer les valeurs d'une propriété, l'utilité des propriétés dynamiques et d'autres éléments. Par la suite, nous verrons l'intérêt des propriétés du point de vue des méta-objets.

6.4.1. Comment définir une propriété ?

Une propriété peut être définie si et seulement si la classe concernée profite du système de méta-objets. La définition d'une telle chose s'effectue à l'aide de la macro Q_PROPERTY() dont le prototype fourni par la documentation est le suivant :

```
Q_PROPERTY(type name
                READ getFunction
                [WRITE setFunction]
                [RESET resetFunction]
                [NOTIFY notifySignal]
                [DESIGNABLE bool]
                [SCRIPTABLE bool]
                [STORED bool]
                [USER bool]
                [CONSTANT]
                [FINAL])
```

On peut donc considérer une déclaration de propriété comme celle-ci :

```
Q_PROPERTY(bool myProperty READ getValue WRITE
setValue);
```

READ et WRITE sont en réalité des fonctions d'accès, respectivement associées à l'accès en lecture et en écriture. Elles permettent respectivement comme leur nom l'indique de récupérer la valeur de la propriété et de la modifier. Ces deux fonctions doivent respecter des règles :

- la fonction READ doit être du type de la propriété et ne doit pas prendre d'argument ;
- la fonction WRITE doit être de type void et doit prendre un seul argument du type de la propriété.

Ici, la fonction getValue devra donc être de type bool et ne prendre aucun argument. Quant à elle, la fonction setValue devra être de type void et prendre un argument de type bool.

```
class MyWidget : public QWidget
{
        Q_OBJECT
        Q_PROPERTY(bool myProperty READ getValue
WRITE setValue);

        public:
        MyWidget();
        void setValue(bool);
        bool getValue() const;

        private:
        QVariant value;
};
```

Le QVariant value, qui aurait d'ailleurs pu être tout simplement un bool, a pour but de prendre la valeur de la propriété myProperty dans les implémentations des fonctions setValue et getValue. Je vous présente ces implémentations, en insistant sur le fait qu'il est parfaitement possible de procéder différemment :

```
void MyWidget::setValue(bool v)
{
        value = QVariant(v);
}
bool MyWidget::getValue() const
{
        return qvariant_cast<bool>(value);
}
```

En sachant que la fonction property, avec comme argument le nom de la propriété (ici, myProperty) retourne un QVariant de la valeur de la propriété, il est possible de s'interroger sur l'utilité de la variable value. Il se trouve que la fonction property se sert de la fonction READ (quand elle est présente) pour retourner une valeur. Si la fonction READ s'appuyait sur la fonction property, une boucle infinie serait créée, d'où l'intérêt de value.

6.4.2. Changer la valeur d'une propriété

À moins que la propriété ait pour but d'être constante (donc à moins qu'il y ait l'attribut CONSTANT dans la déclaration de la propriété), il peut être intéressant de

pouvoir changer la valeur d'une propriété.

Il existe deux méthodes pour changer la valeur d'une propriété :

- utiliser la fonction setProperty avec deux arguments, respectivement le nom de la propriété et sa nouvelle valeur ;
- utiliser tout simplement la fonction WRITE, quand elle existe, en passant en argument la nouvelle valeur.

```
MyWidget widget;
widget.setProperty("myProperty", true);
widget.setValue(true);
```

La fonction setProperty retourne un booléen du fait que l'assignation d'une nouvelle valeur ait réussi ou échoué.

Parfois, le programmeur a besoin de savoir quand la valeur d'une propriété change et donc, de récupérer un signal à chaque modification. C'est par exemple le cas de la propriété text de QLineEdit qui a pour signal de notification textChanged, avec pour argument un QString contenant la nouvelle valeur.

```
Q_PROPERTY(bool myProperty READ getValue WRITE
setValue NOTIFY valueChanged);
// ...
signals:
void valueChanged(bool);
```

Le signal valueChanged doit être émis dans la fonction WRITE lors de la modification de la propriété :

```
void MyWidget::setValue(bool v)
{
        value = QVariant(v);
        emit valueChanged(v);
}
```

Il faut toutefois noter que certaines propriétés n'ont pas de signal de notification, mais qu'il existe tout de même des signaux en rapport. Par exemple, les signaux pressed et clicked de QAbstractButton ne sont pas appelés lorsque la valeur de la propriété down change.

6.4.3. Les propriétés dynamiques

Les propriétés dynamiques constituent en elles-mêmes de bonnes raisons de se servir des propriétés dans un projet. Le principal avantage fourni par les propriétés dynamiques est qu'elles peuvent être ajoutées *pendant* l'exécution. La documentation informe les utilisateurs de propriétés dynamiques qu'elles sont ajoutées à l'objet, et non au méta-objet, ce qui a pour conséquence que la propriété ne peut être retirée par invalidation depuis une instance.

L'ajout d'une propriété dynamique se fait par un appel de la fonction setProperty, en passant un nom de propriété n'existant pas déjà. Toutefois, cette fonction retournera forcément false, même si l'assignation de la valeur à la nouvelle propriété dynamique a été un succès.

Pour plus d'informations sur les propriétés, veuillez vous référer à cette page (Lien62).

6.5. L'introspection

Un des grands avantages du système de méta-objets de Qt est de fournir un système d'introspection. L'introspection est la capacité d'un programme à s'examiner, à s'étudier. Pour être plus précis, il est possible de dire que le RTTI (Lien57) constitue en lui-même un sous-ensemble de l'introspection.

Le RTTI (soit *Run Time Type Information* en anglais et donc *Informations de Type disponibles à l'Exécution* en français) peut être considéré comme un mécanisme permettant comme son nom l'indique de déterminer le type des objets durant l'exécution. Dans le cas du C++, le RTTI est associé à l'opérateur typeid et à la classe type_info, donc au header <typeinfo>. Pour plus d'informations concernant le RTTI sur ce langage, veuillez consulter cet article (Lien57).

Avec Qt, les informations de type peuvent être gérées de la même manière qu'en C++ mais bénéficient d'un support plus efficace. Ce support est basé sur les méta-objets et sur QObject, la classe de base de la quasi-totalité des classes de Qt. Il est donc principalement géré par le compilateur de méta-objets (moc).

Dans le cas de Qt, l'introspection permet, par exemple, de connaître le nom, le type de retour, le type d'accès, etc. d'une méthode, d'un constructeur ou autres. Cet exemple n'a pas été choisi au hasard. En effet, un exemple sera fourni durant cette partie, concernant la récupération des informations des méthodes et des constructeurs d'une classe. Comment utiliser l'introspection ? Cette question est celle à laquelle il me faudra répondre dans cette partie.

Comme vous avez probablement dû le comprendre, chaque classe dérivée de QObject contenant la macro Q_OBJECT possède son propre méta-objet, soit une seule et unique instance de QMetaObject. Cette instance peut être récupérée à l'aide de la fonction metaObject() de QObject. L'instance récupérée est logiquement constante car les méta-objets ne peuvent pas subir directement de traitement intercessif, c'est-à-dire qu'on ne peut pas modifier, par exemple, le nom d'une classe par le biais de son méta-objet. Cette fonction nécessite la présence d'une instance de la classe concernée. Dans le cas où aucune instance ne serait accessible pour utiliser la fonction metaObject(), la variable staticMetaObject peut être utilisée (exemple : QDialog::staticMetaObject). Toutefois, son cas ne sera pas traité ici.

```
const QMetaObject *_metaObject = metaObject();
```

A partir d'une instance de QMetaObject, on peut utiliser un bon nombre de fonctions informatives, telles que les fonctions method(), constructorCount(), access() et bien d'autres.

```
QString name(_metaObject->className());
// name contient le nom de la classe dont le
méta-objet est _metaObject.
```

Voici un fragment de code dans lequel on stocke toutes les méthodes et tous les constructeurs de la classe concernée dans un tableau, en indiquant la « signature » de la méthode, son type de retour (bool, QString, etc.), son type

de fonction (signal, slot, méthode ou constructeur) et son type d'accès (privé, protégé ou public).

```
QTableWidget *table = new
QTableWidget(this);
table->setColumnCount(3);
table->setHorizontalHeaderItem(0, new
QTableWidgetItem("Type d'accès"));
table->setHorizontalHeaderItem(1, new
QTableWidgetItem("Type de méthode"));
table->setHorizontalHeaderItem(2, new
QTableWidgetItem("Type de fonction"));

table->setRowCount(_metaObject->methodCount() +
_metaObject->constructorCount());
for(int i = 0; i < _metaObject->methodCount(); i+
+)
{
        table->setVerticalHeaderItem(i, new
QTableWidgetItem(_metaObject-
>method(i).signature()));
        table->setItem(i, 2, new
QTableWidgetItem(_metaObject-
>method(i).typeName()));


        if(_metaObject->method(i).access()
==      QMetaMethod::Private)   table->setItem(i,
0, new QTableWidgetItem("Privé"));
        if(_metaObject->method(i).access()
==      QMetaMethod::Protected) table->setItem(i,
0, new QTableWidgetItem("Protégé"));
        if(_metaObject->method(i).access()
==      QMetaMethod::Public)    table->setItem(i,
0, new QTableWidgetItem("Public"));
        if(_metaObject->method(i).methodType()
==      QMetaMethod::Signal)    table->setItem(i,
1, new QTableWidgetItem("Signal"));
        if(_metaObject->method(i).methodType()
==      QMetaMethod::Slot)      table->setItem(i,
1, new QTableWidgetItem("Slot"));
        if(_metaObject->method(i).methodType()
==      QMetaMethod::Method)    table->setItem(i,
1, new QTableWidgetItem("Méthode"));
}
for(int tmp = _metaObject->methodCount(); tmp <
_metaObject->methodCount() + _metaObject-
>constructorCount(); tmp++)
{
        int i = tmp - _metaObject->methodCount();
        table->setItem(tmp, 1, new
QTableWidgetItem("Constructeur"));
        table->setItem(tmp, 2, new
QTableWidgetItem(_metaObject-
>constructor(i).typeName()));


        table->setVerticalHeaderItem(tmp, new
QTableWidgetItem(_metaObject-
>constructor(i).signature()));
        if(_metaObject->constructor(i).access()
== QMetaMethod::Private)        table-
>setItem(tmp, 0, new QTableWidgetItem("Privé"));
        if(_metaObject->constructor(i).access()
== QMetaMethod::Protected)      table-
>setItem(tmp, 0, new
QTableWidgetItem("Protégé"));
        if(_metaObject->constructor(i).access()
== QMetaMethod::Public)         table-
>setItem(tmp, 0, new QTableWidgetItem("Public"));
}
```

À l'exécution, il est possible de voir de diverses méthodes, selon la classe concernée. On constate d'ailleurs que les signaux et les slots sont aussi stockés dans le méta-objet. C'est avec ce code que l'on peut comprendre une chose : les méta-objets ne disposent que des fonctions qui leur sont fournies par le moc. Un slot par exemple est ajouté au méta-objet, tandis qu'une méthode publique ne l'est pas sans la présence d'une macro du type de Q_INVOKABLE.

Voici un petit tableau contenant les principales macros utiles pour le recensement de méthodes par le moc :

Macro	Description rapide
Q_INVOKABLE	Permet au méta-objet d'appeler la méthode.
Q_SIGNAL	Marque la méthode en tant que signal.
Q_SLOT	Marque la méthode en tant que slot.

Il existe bien entendu d'autres macros, comme par exemple Q_SCRIPTABLE.

6.6. Les méta-objets

Les méta-objets constituent le point central de l'article et nous les avons déjà abordés théoriquement dans les parties situées avant celle-ci. La partie précédente envisage une première manière de coder avec les méta-objets dans un but introspectif, ce qui correspond à une petite partie de ce que le système de méta-objet fournit.

Notez que cette partie n'a pas pour but de recenser toutes les macros mises à disposition par le système de méta-objets. Elle présentera les différents dispositifs mis à disposition sous la forme présentation/exemple.

6.6.1. Les superclasses

On nomme superclasse d'un objet sa classe parente. Les superclasses sont utilisées par QMetaObject pour toutes ses fonctions retournant un offset, par exemple par classInfoOffset().

Il est possible de récupérer le méta-objet de la superclasse d'un objet héritant directement ou non de QObject en appelant la fonction superClass() de QMetaObject. Si cet objet profite d'un héritage multiple, superClass() retournera le méta-objet de la première classe dont il hérite, dont forcément une classe héritant de QObject. Si l'objet n'a pas de superclasse, superClass() retourne 0.

Exemple d'utilisation :

```
const QMetaObject *_metaObject =
metaObject()->superClass();
while(_metaObject != 0)
{
        qDebug() << _metaObject->className();
        _metaObject = _metaObject->superClass();
}
```

Étant en haut de la hiérarchie des classes de Qt, QObject sera le dernier nom de classe affiché par qDebug().

6.6.2. Fonctions d'index

Les fonctions d'index (soit dans Qt 4.6 indexOfConstructor(), indexOfMethod(), indexOfEnumerator(), indexOfProperty(), indexOfSlot(),

etc.) sont des fonctions permettant de récupérer l'index d'un constructeur, d'une méthode, etc. à partir de son nom, ou bien -1 si aucun index n'a été trouvé. La documentation précise l'utilité des fonctions d'index en disant que lors de la connexion de signaux à un slot, Qt utilise la fonction indexOfMethod(). Les paramètres passés doivent être normalisés, donc par exemple être mis sous une forme possédant le minimum d'espaces. Il faut donc qu'ils soient passés sous la fonction normalizedSignature().

Exemple d'utilisation dans une classe possédant un slot dont le prototype est myFunction(QString) :

```
const QMetaObject *_metaObject =
metaObject();
qDebug() << _metaObject-
>indexOfMethod(_metaObject-
>normalizedSignature("myFunction(QString)"));
```

6.6.3. L'appel de méthodes

Le système de méta-objets permet aussi d'appeler des méthodes par le biais de la fonction statique invokeMethod() et de ses fonctions de convenance. Voici le prototype complet de cette fonction :

```
bool QMetaObject::invokeMethod(QObject
*obj,
  const char *member,
  Qt::ConnectionType type,
  QGenericReturnArgument ret,
  QGenericArgument val0 = QGenericArgument(0),
  QGenericArgument val1 = QGenericArgument(),
  //...,
  QGenericArgument val9 = QgenericArgument());
```

Selon le *type* de connexion, cette fonction appellera la méthode renseignée par *member* (qui doit être recensée par le moc), de l'objet *obj*, avec les arguments renseignés, et retournera un booléen du fait que l'appel ait réussi ou non.

Les arguments *val0*, *val1*, etc. passés doivent être sous forme de QGenericArgument. Nous utilisons donc une macro nommée Q_ARG, où l'on renseigne le type puis la valeur de l'argument souhaité, qui retourne un objet de ce type.

Exemple d'utilisation dans une classe possédant un slot dont le prototype est myFunction(QString) :

```
const QMetaObject *_metaObject =
metaObject();
qDebug() << _metaObject->invokeMethod(this,
"myFunction", Qt::DirectConnection,
Q_ARG(QString, "Qt is good !"));
```

6.6.4. La création d'instances

Tout comme il peut appeler des méthodes, le système de méta-objets permet aussi de créer des instances. Cela peut être fait à l'aide de la fonction newInstance() de QMetaObject, dont le prototype est le suivant :

```
QObject*
QMetaObject::newInstance(QGenericArgume
nt val0 = QGenericArgument(0),
                                        QGenericA
rgument val1 = QGenericArgument(),
                                        ...,
                                        QGenericA
rgument val9 = QGenericArgument()) const
```

Cette fonction permet de créer une instance de l'objet auquel le méta-objet appartient. Lors de l'appel de cette fonction, Qt va tenter de trouver un prototype correspondant aux arguments donnés et va l'appeler. Il va de soi que les constructeurs concernés doivent pouvoir être appelés, donc être déclarés avec la macro Q_INVOKABLE pour être recensés par le moc. Si aucun prototype ne correspond, 0 est retourné.

Comme pour l'appel d'une méthode par le système de méta-objets, *val0*, *val1*, etc. passés doivent être sous forme de QGenericArgument, donc être renseignés avec la macro Q_ARG.

Exemple d'utilisation dans le cadre d'une classe nommée MyWidget, dérivée de QWidget, dont le prototype est Q_INVOKABLE MyWidget(QObject *parent = 0) :

```
const QMetaObject *_metaObject =
metaObject();
MyWidget *newObject = qobject_cast<MyWidget
*>(_metaObject->newInstance(Q_ARG(QObject *,
this)));
if(newObject) newObject->show();
```

6.6.5. Les propriétés

Si nous ne considérions pas l'aspect des propriétés du côté des méta-objets, il nous aurait été possible de voir les propriétés comme un outil peu important, peu utile. Or, la déclaration d'une propriété à l'aide de la macro Q_PROPERTY engendre son recensement par le moc, qui va l'ajouter dans le méta-objet.

La fonction property() d'une instance de QObject retourne la valeur de la propriété passée en argument. Toutefois, passer l'index de la propriété dans la fonction property() d'une instance de QMetaObject retournera un QMetaProperty.

Voici un exemple retournant le nom de toutes les propriétés d'un méta-objet :

```
const QMetaObject *_metaObject =
metaObject();
for(int i = 0; i < _metaObject->propertyCount();
i++)
qDebug() << _metaObject->property(i).name();
```

QMetaProperty possède de multiples méthodes pouvant, par exemple, retourner un QMetaMethod du signal de notification. Pour plus d'informations, veuillez consulter cette page (Lien63).

7. Conclusion

Les méta-objets sont donc bien plus complexes et plus utiles qu'on ne pourrait le croire, même s'ils ne correspondent pas à un outil qu'un codeur d'interface graphique utilise fréquemment dans son propre code. Il les utilise, oui, mais ne s'en occupe a priori pas. Il faut noter que cet article présente une partie importante du système, mais n'est pas non plus exhaustif. Si vous souhaitez en apprendre plus sur le sujet, vous pouvez toujours consulter la documentation, par exemple.

Retrouvez l'article de Louis du Verdier en ligne : Lien64

Objective-C 2.0
Le langage de programmation iPhone et Cocoa sur Mac Os X

Ce Guide de survie est l'outil indispensable pour maîtriser Objective-C, le langage utilisé pour écrire les applications natives Mac OS X et iPhone. Vous y trouverez les bases d'Objective-C, ainsi que tout ce qu'il faut savoir pour bien gérer la mémoire, comprendre le système de notification et d'événements, migrer de la version 1.0 à 2.0, réaliser des tests unitaires et améliorer la qualité du code. Si vous venez d'autres langages, comme Java, C++, C# ou Python, il vous aidera à assimiler rapidement les spécificités d'Objective-C.

Critique du livre par Aurélien Gaymay

Ce livre est un guide de survie pour les développeurs Objective-C connaissant déjà le langage de programmation. Il est très bien détaillé et permet un passage facile de l'Objective-C 1.0 à la version 2.0. Ce n'est pas dans ce livre que vous allez apprendre à développer des applications en Objective-C (iPhone ou Mac), car il faut quand même un minimum de bases dans le domaine. Mais avec de bonnes connaissances en C/C++, ça peut le faire.

Le livre est un bon aide-mémoire pour tous les développeurs Cocoa, et en plus, il est facile et agréable à lire.

Critique du livre par Vincent Brabant

J'ai un problème avec ce livre, car l'intitulé exact de celui-ci est ceci :

Le Guide de survie
Objective-C 2.0
Le langage de programmation iPhone et Cocoa sur Mac Os X

Or il est très peu fait mention de l'iPhone dans ce livre, et encore moins de Cocoa.

Les points abordés comme la gestion de la mémoire, les test unitaires, la gestion des exceptions... sont très bien abordés.

Mais par contre, du fait du sous-titre le langage de programmation iPhone et Cocoa sur Mac Os X, on croit qu'on va également apprendre des bonnes pratiques pour le développement sous iPhone ou avec Cocoa. Mais ce n'est jamais le cas.

Je ne sais même pas, après avoir lu ce livre, comment compiler une classe Objective-C, comment organiser mes sources, comment écrire une boucle en Objective-C.

(j'ai cru comprendre qu'il n'y a pas de notion de package en Objective-C, comme c'est le cas en Java par exemple.
Mais pas un mot sur comment faire alors pour éviter un clash entre les classes. Comment être sûr que l'import de la classe Xyz fait bien référence à ma classe Xyz que j'ai écrite et non à celle écrite par quelqu'un d'autre.
Bref, vous voyez que même après avoir lu le livre, le doute n'est pas levé. Or c'est tout de même pas une bête question, non ?)

Et je ne sais toujours pas comment écrire tout simplement un Hello World en Objective-C.

Pour un guide de survie, c'est tout de même assez spécial, non.

Je veux dire par là qu'il ne s'adresse pas du tout à des débutants, mais à des personnes qui ont déjà une maîtrise de Objective C, que ce soit 1.0 ou 2.0 et qui veulent en savoir plus sur Objective-C lui-même.

Mais qu'elles n'espèrent pas avoir de l'aide pour développer leur première application pour iPhone par exemple.
Pas un mot sur CocoaTouch par exemple.
Et n'espérez pas non plus pouvoir écrire votre premier petit programme en Objective C sous Mac Os X après avoir lu ce livre.

De plus, il y est fait un amalgame entre Objective-C et Cocoa, qui embrouille vraiment le lecteur.

Pour résumer, les sujets qu'il aborde, il les aborde très bien (sauf la 1re section), mais je m'attendais à ce qu'il aborde d'autres points, ce qu'il ne fait jamais. Donc oui, je suis déçu.

Sa cible n'est clairement pas les débutants dans ce langage, mais déjà des experts. **Mais est-ce que les experts y apprendront quelque chose ?**
Oui. Les nouveautés introduites dans Obective-C 2.0 pour ceux qui ne connaissaient que Objective C 1.0.

La gestion de mémoire est ma section préférée. C'est vraiment ce que je m'attendais à trouver dans un tel livre :

- les bonnes pratiques comme la gestion d'erreur ;
- comment écrire des tests unitaires.

Vraiment, ces parties là sont d'un bon niveau.

Difficile de coter ce livre, car **il mérite une bonne cote pour les sujets qu'il aborde**, mais il ne mérite pas une bonne cote car il ne répond pas à nos attentes (attentes provoquées par le sous-titre du livre mentionnant iPhone et Cocoa).

Je tenais à préciser; pour terminer, que je n'ai jamais développé en Objective-C de ma vie. **La lecture de ce**

livre m'a appris pas mal de choses sur l'Objective-C, mais rien qui me permette de dire que je me sens maintenant à l'aise avec ce langage.
Rien qui me permettra d'écrire mon premier petit programme sur iPhone et/ou Mac (même un Hello World).

Et je ne me souviens pas d'avoir vu des classes typiques à Cocoa dans ce bouquin.

Retrouvez cette critique de livre sur la page livres Mac : Lien65

Les derniers tutoriels et articles

Apple Event Avril 2010 : Apple dévoile l'iPhone OS 4.0

Ce 8 avril 2010, Apple a présenté certaines des nouveautés de l'iPhone OS 4.0 aux journalistes lors d'un Apple Event.
Voici un résumé de ce qui a été dit durant la keynote, mais aussi ce qui n'a pas été dit durant cette même keynote.

1. Introduction

Après le lancement de l'iPad durant le weekend du 3 avril, Apple n'a pas tardé à levé le voile sur la prochaine version de l'iPhone OS.

Mais avant de parler de l'iPhone OS proprement dit, Steve Jobs, aux commandes de cette keynote, est d'abord revenu sur le lancement de l'iPad.

2. Quelques chiffres

2.1. Concernant l'iPad

Alors qu'Apple a vendu 300.000 iPad le samedi, il s'en est maintenant vendu 450.000.

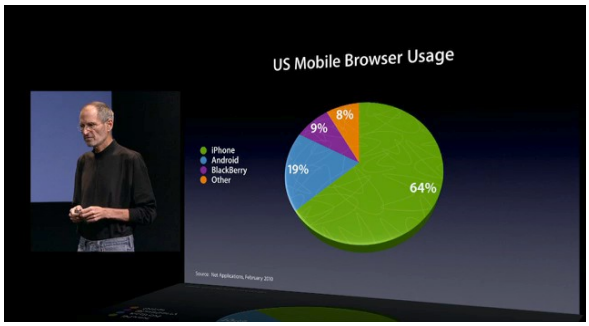
Les heureux possesseurs d'un iPad avaient le 1er jour téléchargé 250.000 iBook.
Ils en ont maintenant téléchargés 600.000. Soit plus d'iBooks que d'iPad vendus. Ce qui n'était pas le cas le 1er jour.

Pour ce qui est des applications, 1.000.000 d'applications pour iPad avaient été téléchargées le 1er jour.
C'était déjà plus de 3.500.000 applications qui avaient été téléchargées depuis.
Un peu plus de 7 applications par iPad donc.

Il faut savoir qu'il existe actuellement, parmi les 185.000 applications que l'on retrouve sur l'AppStore, 3.500 applications dédiées à l'iPad.
Et qu'il y a eu 4 **milliards** d'applications téléchargées depuis la création de l'AppStore.

2.2. Concernant l'iPhone

Apple a obtenu le prix 2010 de JD Power "Highest Customer Satisfaction" pour l'iPhone.
Prix 2010 qui vient se rajouter au même prix obtenu en 2008 et en 2009.



En février 2010, l'iPhone représentait 64% du trafic des navigateurs mobiles aux États-Unis. Soit presque le double de tous les autres acteurs réunis, Android occupant déjà la 2e place avec 19% et BlackBerry la 3e place avec 9%.

Lorsqu'on regarde ce camembert, on a l'impression de voir un PacMan vert en train de manger des fantômes bleus, mauves et oranges.

Apple a vendu jusqu'à présent 50.000.000 d'iPhone. Et si on rajoute les iPod Touch, cela monte jusque 85.000.000.

3. iPhone OS 4.0 : les grandes lignes

L'iPhone OS 4.0 sera disponible pour les utilisateurs cet été.
Mais les développeurs peuvent déjà, dès aujourd'hui, télécharger une version preview.

L'iPhone OS 4.0 vient avec 1500 nouvelles API. Il est bien évidemment impossible de toutes les citer ici, mais voici déjà toutes celles qui sont mentionnées sur ce slide :

Quelques unes des 1500 nouvelles API
- Date Data Detectors,
- Block Based animation
- Automated testing
- Performance profiling tools
- Accelerate
- Embed PDF metadata
- Draggable map annotations
- Full map overlays
- Power Analysis tools
- Carrier information
- ICC Profiles
- Calendar Access
- Address Data Detectors
- iPod remote control accessories
- In-App SMS
- Regular Expression matching
- Date formatters
- Photo Library Acces
- Image I/O
- Half Curl Page Transition
- Quick Look
- Package Based documents
- Call event notifications
- Full access to still and video camera data

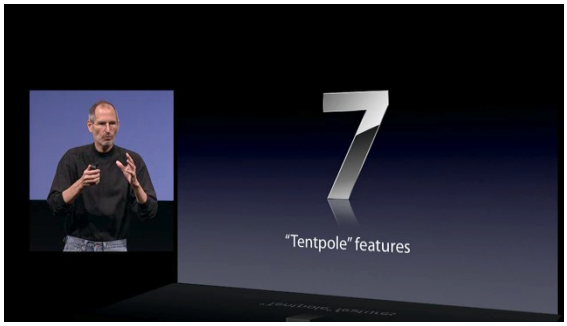
Ce qui est fort intéressant pour nous, développeurs, ce sont les API permettant l'automatisation des tests, mais aussi les outils de profiling des performances.

A coté de ces 1500 nouvelles API mises à disposition des développeurs, l'iPhone OS 4 fournira 100 nouvelles fonctionnalités aux utilisateurs.
A nouveau, impossible de toutes les citer ici, mais voici déjà toutes celles qui sont mentionnées sur ce slide :

Quelques unes des 100 nouvelles fonctionnalités
- Birthday calendar
- Rotate photo
- Recent Web Searches
- Gift Apps
- Persistent Wi-Fi
- Cell data only setting
- Spell Check
- Larger font for Mail, SMS & Alerts
- CalDAV invitations
- Bluetooth keyboards
- Choose image size in Mail messages
- Web Search suggestions
- Edit from Outbox
- Create playlists
- Top Hit in search
- Sync IMAP Notes
- 5x digital zoom
- Nested playlist
- CardDAV
- Tap to focus video
- Upload workouts to Nike+
- Places in Photos
- iPod Out
- Home Screen wallpaper
- Search SMS/MMS messages
- Wake on wireless
- File & delete Mail search results

Ceci est mon avis tout à fait personnel, mais proposer un zoom numérique de 5x sur les iPhone actuels n'est pas d'une grande utilité. Par contre, cette fonctionnalité prend tout son sens si la rumeur concernant le nouvel iPhone qui serait dévoilé à l'occasion du WWDC 2010 s'avere fondée. Cette rumeur parle d'un iPhone HD. HD comme dans High Definition, ce qui me laisse penser que cet iPhone devrait avoir un meilleur capteur que celui qu'on trouve pour le moment sur l'iPhone, et pour lequel un Zoom 5x aurait une quelconque utilité .

4. iPhone OS 4.0 : les sept grosses nouveautés mise en avant par Apple



Apple a décidé ce soir de mettre 7 grosses nouveautés, sur la centaine que contiendra l'iPhone OS 4.0.

4.1. Multitasking

La première de ces 7 grosses nouveautés est l'apparition du multi-tâche
Mais Apple n'a pas fait les choses tout bêtement, en permettant à toutes les applications de tourner en tâche de fond, vidant completement votre batterie ou ralentissant votre iPhone.

Les gens se sont plaint de l'absence du Multitasking tout comme ils se sont plaint de l'absence du copier/coller lors des premières versions de l'iPhone.
Pour le copier/coller, il a fallu du temps, mais la solution finalement proposée par Apple est agréable.

Pour le multitasking, Apple a également pris le temps, car, selon Steve Jobs, Apple voulait proposer le meilleur. Une solution qui tient la route. S'ils avaient implémenté le multitasking comme d'habitude, la batterie se serait vidée rapidement, et les performances auraient chuté de façon dramatique, laissant un gout amer chez l'utilisateur.

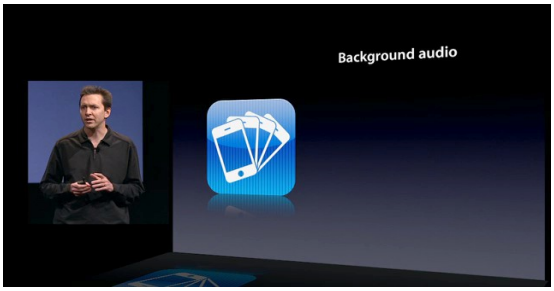
La solution proposée par Apple devrait séduire les utilisateurs, mais également les développeurs, même si leur application ne tournera pas vraiment en parallele, comme c'est actuelement le cas sur les OS de bureaux que sont Mac OS X, Linux ou Windows, pour ne citer que ces trois là.

Au lieu de cela Apple met à disposition des développeurs toute une série d'API qui donneront l'illusion à l'utilisateur que votre application continue à tourner en tâche de fond, alors qu'une partie seulement tourne réellement.
Voyons maintenant quelques unes de ces possibilités données aux développeurs.

Les 7 services de Multi-Tasking proposés aux développeurs

- BackGround Audio
- Voice Over IP
- BackGround Location
- Push Notifications
- Local Notifications
- Task completion
- Fast App Switching

4.1.1. BackGround Audio



Le Service de ***BackGround Audio*** permettra aux utilisateurs de l'iPhone de continuer à écouter de la musique jouée par une application après l'avoir quitté pour ouvrir d'autres applications.

Ils ont pris l'exemple de Pandora. Qui est une application vous permettant d'écouter la radio sur internet.
Jusqu'à présent, si vous vouliez écouter la radio sur Internet pendant tout votre voyage en train, par exemple, vous ne pouviez rien faire d'autre avec votre GSM pendant ce temps là. Alors que vous pouvez écoutez de la musique provenant d'iTunes, tout en lisant vos mails.

Avec ***BackGround Audio***, vous allez pouvoir quitter l'application Pandora, ou n'importe quelle autre application utilisant ce service pour allez, par exemple, lire vos mails, ou surfer sur le net, tout en continuant à écouter la musique qui était jouée par Pandora.

Avec la version de Pandora présentée lors de cette keynote, vous pourrez même vous rendre, depuis Pandora, vers l'iTunes Store pour y acheter la musique que vous êtes toujours en train d'écouter, alors que vous avez quitter l'application Pandora.

Rien que ce service là tout seul peut déjà convaincre n'importe quel utilisateur d'iPhone à mettre à jour son iPhone OS vers la version 4 dès que possible.

4.1.2. Voice Over IP

La deuxième fonction proposée est celle de ***Voice Over IP***

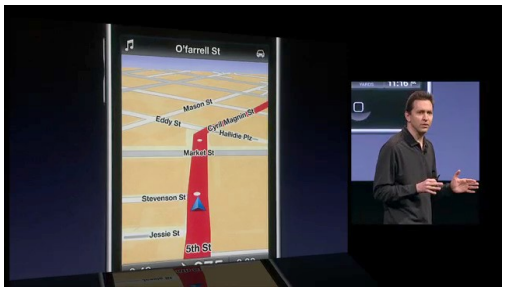
Ainsi, vous pourrez lancer Skype, commencer à discuter avec une personne sur skype, puis, tout en continuant la discussion avec cette personne, fermer Skype pour effectuer une recherche sur le net et donner à votre correspondant Skype le résultat de cette recheche.

Et vous pouvez même, alors que vous n'êtes pas dans Skype à ce moment là, recevoir une notification vous indiquant qu'un correspond Skype vous appelle.

Ce service là rends Skype vraiment utilisable sur iPhone. Ce qui, avouons le, n'était pas vraiment le cas jusqu'à présent vu les limitations dues à l'absence du multi-threading.

4.1.3. BackGround Location

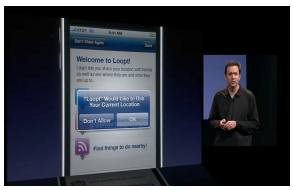
Pour le service de BackGround Location, rien de tel que de prendre l'exemple de Tom-Tom.

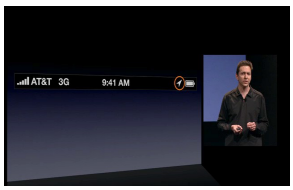


Imaginez que vous lancer Tom-Tom pour avoir le guidage.

Durant la keynote, Scott a également rappelé que pour Apple, la vie privée était quelque chose de sérieux.

Scott a donc proposé 3 nouveautés qui va mieux renseigner l'utilisateur sur quand sa géolocation est tracée, et par qui et quand.





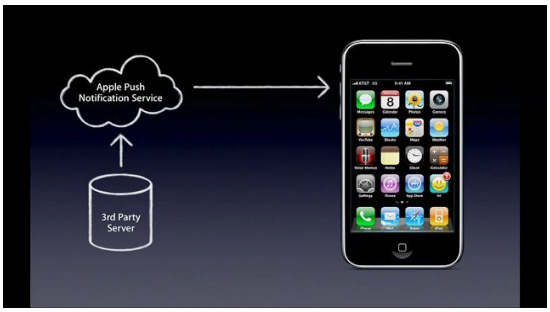
Maintenant, chaque fois qu'une application est en train de vous positionner une icône apparaitra à coté de l'heure.

Vous avez également indiquer, pour chacune des applications, si vous êtes d'accord qu'elle vous positionne ou pas.

Et finalement, dans cette liste d'application, la même icone apparaitra à coté du nom de l'icone si celle-ci vous a positionné dans les dernières 24 heures.

4.1.4. Push Notification

Une notification est transmise par un serveur tier vers le service de Push Notification d'Apple. Celui relaye cette notification à l'iPhone.



Ce système de Push Notification est en place depuis 9 mois maintenant.
Et rencontre un certains succès puisque c'est pas moins de 10 milliards de notifications qui ont ainsi été transmises sur les iPhones, transitant par les serveurs Apple.

Le principal avantage du système de Push Notification tel que présenté par Apple est qu'il n'y a qu'une seule connection établie entre le service d'Apple et l'iPhone. Alors qu'il aurait fallu autant de connections que de serveurs tiers en l'absence de ce service.

Mais cela génère pas mal de traffic. Par exemple, vous configuriez une application sur votre iPhone pour vous envoyer une notification à un moment donné. L'application contactait son serveur tier pour lui dire quand il devait envoyer la notification.

Et au moment donné, le serveur tiers contactait le service de Push Notification qui contactait votre iPhone qui affichait la notification sur votre écran.

Ce scénario là va pouvoir être simplifié avec l'apparition du système suivant.

4.1.5. Local Notification

Le Système de Local Notification est identique à celui du Push Notification, excepté que tout se passe uniquement sur votre iPhone.
Aucun serveur tiers n'intervient dans ce scénario.

Par exemple, une application de style Guide TV enregsitra une notification pour une heure bien précise
Et celle-ci s'affichera au moment précis.

Cela peut paraitre simple et évident, mais cela nécessitait l'introduction du multi-tache qui permet à un deamon de tourner en tâche de fond et qui scrute le calendrier de l'iPhone pour afficher une notification au moment demandé.

4.1.6. Task Completion

L'exemple repris par Scott est encore une fois bien parlant.

Vous avez pris une photo avec votre iPhone et voulez la publier sur Flickr.

Si vous quittiez l'application Flickr avant que l'envoi de la photo soit terminé, et bien l'envoi était terminé. Et la photo n'était jamais publiée sur Flickr.

Maintenant, vous pouvez vous rendre dans l'application Flickr, lui demander l'envoi de votre photo, pour tout de suite quitter Flickr et passer à autre chose.

Grâce au système de Task Completion, l'application Flickr continuera à envoyer la photo sur le serveur de Flickr.

4.1.7. Fast App Switching

Lorsque vous désirez basculer d'une application à l'autre, il faut que cela se fasse quasi instantanément.

C'est ici qu'intervient le Service Fast App Switching. Lorsque l'utilisateur "quitte" une application, celle-ci fait appel à ce service, qui va "geler" en mémoire l'application.

Tout son contenu, son état, ... est comme "gelé".
L'application n'utilise plus de temps cpu.
Et sera rétablit dans cet état là lorsque l'utilisateur reviendra sur cette application.

4.2. Folders : Les dossiers

Les dossiers étaient également quelque chose de fort demandé par les utilisateurs. Et donc Apple l'a finalement implémenté.

Les utilisateurs d'iPhone téléchargent beaucoup d'applications et devaient jusqu'à présent se promener

d'une page à l'autre pour retrouver l'application qu'il désirait lancer.

Il fallait donc un meilleur moyen d'organiser les applications et de les retrouver plus facilement.

Et la réponse, c'est les dossiers.

Et la création d'un folder se fait de façon très simple. Vous prenez une application, comme si vous vous apprêtiez à la déplacer et vous la déposer sur une autre application : un dossier est créé.

Il y a même un nom de dossier qui vous est proposé par défaut.
Ce nom de dossier reprendra la catégorie commune aux deux applications.

Vous pouvez bien évidemment changer le nom qui vous est proposé par défaut.

Ainsi, grâce aux dossiers qui peuvent recevoir 12 applications, il est maintenant possible d'avoir plus de 2000 applications disponibles. (9 pages * 20 dossiers * 12 applications = 2160 applications)

4.3. iBooks

iBooks, l'application phare d'Apple pour mettre en avant l'iPad est maintenant disponible sur iPhone.

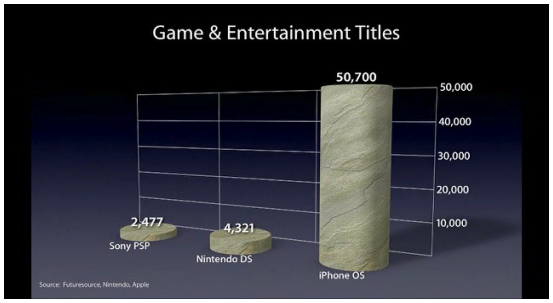
Apple promet qu'on pourra ainsi commencer la lecture d'un livre sur l'iPad, la poursuivre sur l'iPhone, et la terminer sur l'iPad.

Les pages et les signets seront synchronisés entre les différents appareils.

4.4. Game Center

Les jeux ont pris une grande place sur l'iPhone et l'iPod Touch.

En fait, parmi les 180.000 applications disponibles sur l'App Store, 50.000 sont des jeux.



Apple a même oser comparer les 50.000 jeux disponible sous iPhone OS avec les près de 2500 jeux pour Sony PSP et 4300 jeux pour Nitendo DS, ne manquant pas de souligner que cela représente 10 fois plus de jeux.

Prenant conscience de cela, Apple veut aller encore plus loin maintenant en créant un réseau social pour les jeux.

Il sera ainsi possible d'inviter des amis à jouer au jeu auquel on joue, de comparer leur score avec le notre, de voir les meilleurs score, de voir votre progression dans le jeux, ..., ...

Attention que cela n'est qu'une preview, et ne sera pas disponible fin juin avec les autres fonctionnalités de l'iPhone OS 4, mais plus tard dans l'année.

4.5. iAd : Mobile Advertising

C'est à nouveau Steve Jobs qui remonte pour présenter la dernière fonctionnalité qu'est iAd.

Apple a intégré iAd directement dans l'iPhone OS 4.

Beaucoup d'applications disponibles sur l'App Store y sont disponible gratuitement.

Les utilisateurs aiment cela, Apple aime cela, mais cela complique la position des développeurs.

Et donc, les développeurs ont introduit de la publicité dans leurs applications gratuites.

Mais, comme Steve Jobs l'a lui-même dit, de façon très élégante, la plupart de ces publicités " really sucks".

Sur les ordinateurs de bureau, l'endroit où se trouvent les publicités, c'est sur les moteurs de recherche.

Mais sur les iPhone, les utilisateurs ne passent pas leur temps à effectuer des recherches. Ils passent leur temps dans les applications. Et donc, pour les mobiles, le mieux serait d'intégrer la publicité au sein des applications.

C'est ce que propose Apple avec iAd.

D'après Apple, les utilisateurs d'iPhone passe en moyenne 30 minutes dans les applications, par jour.

Steve Jobs vous demande d'imaginer ce que ce serait si

une publicité serait affichées toutes les 3 minutes.

Cela ferait 3 publicités, par appareil mobile, par jour.

En assumant que bientôt il y aura en circulation 100 millions d'appareils mobiles d'Apple (iPhone, iPod Touch, iPad Wi-fi et iPad 3G), cela ferait, au total 1 milliard de publicités affichées, par jour.

On comprends, en lisant ces chiffres de 1 milliard de publicités affichées par jour, tout l'intérêt qu'a Apple de se lancer dans ce marché, qui peut s'avérer très lucratif.

Mais Apple ne veut pas s'arrêter là. La société veut aller plus loin. Elle veut changer la qualité des publicités.

Toujours d'après Steve Jobs, les publicités sur le net sont interactives, mais il leur manque l'émotion. Raison pour laquelle les publicitaires continuent à passer des publicités à la télévision car la télévision permet de jouer sur l'émotion. Mais elle ne permet pas l'interaction.

Avec iAd, Apple veut proposer non seulement ET de l'émotion, ET de l'interaction, mais apporter encore plus d'interaction.

Parce que iAd est directment au coeur de l'iPhone OS, l'utilisateur ne quitte pas vraiment l'application.

Et Steve Jobs de déclarer que pour les développeurs, il sera aisé modifier leur applications pour y inclure iAd. Cela ne prendrait pas plus qu'une après-midi.

Et il finit en déclarant qu'Apple redistribuera 60% des revenus aux développeurs.

Voyons maintenant comment cela va fonctionner, avec des exemples fictifs créés par Apple :

Imaginez que vous êtes dans une application qui vous affiche des news.

En bas de cette application apparait une petite bannière pour Toy Story 3.

Vous cliquez sur la bannière, et la publicité surgit pour envahir tout l'écran

De là, vous avez accès à des bandes-annonces, des petits jeux, les endroits où le film est projeté, des fonds d'écrans pour votre iPhone, …
Et surtout, la proposition de télécharger un jeu depuis l'App Store.

Ces publicités sont réalisées, on s'en doutait en HTML 5. Et certainement pas en Flash.

Steve Jobs a insisté durant tout le temps de la démo, que l'utilisateur peut à tout moment quitter la publicité en cliquant sur la croix situé dans le coin supérieur gauche.

5. Conclusion de la keynote

L'iPhone OS 4 est disponible pour les développeurs aujourd'hui.
Il le sera pour les utilisateurs d'iPhone et iPod Touch cet été.

L'iPhone 3GS et l'iPod Touch 3rd Gen supporteront complètement les nouveautés de l'iPhone OS 4.

L'iPhone 3G et l'iPod Touch 2nd Gen supporteront quelques fonctionnalités de l'iPhone OS 4.
Le grand absent sera le multi-tasking.

Et l'iPhone OS 4 sera disponible pour les utilisateurs d'iPad cet automne.

Retrouvez la suite de l'article de Marcos Ickx en ligne : Lien66

Compte-rendu XP Day Suisse 2010

Ce lundi 29 mars j'ai eu le plaisir d'assister à la deuxième édition de la conférence XP Day Suisse (Lien67), qui se tenait à Genève, tout comme l'année dernière (voir le compte-rendu de Pierre Caboche (Lien68)). Cette conférence bénéficie du soutien de plusieurs sponsors, dont developpez.com (Lien69), et d'une équipe d'organisateurs dynamiques et passionnés. J'ai d'ailleurs trouvé excellente toute l'organisation, que ce soit le choix du lieu, le "time-boxing" des diverses séances, les buffets, les zones de temps réservées aux échanges informels et au réseautage, le vote immédiat en fin de séances par gommettes de couleurs, le bon sac en toile de jute durable avec comme "goodies" juste le jeu de cartes de l'alchimiste agile (Lien70). Félicitations donc aux organisateurs qui nous ont permis de passer une excellente journée.

Un petit bémol toutefois sur le fait qu'il faut absolument payer l'inscription par un virement international (enfin, pour les Français) : un de mes collègues a eu la désagréable surprise de découvrir que sa banque prenait en commissions diverses la moitié du prix de l'inscription ! Toujours pour les Français, sachez que nos amis Suisses sont moins fanas de la carte de crédit que nous, et qu'il vaut mieux avoir de l'argent suisse pour régler le parking situé sous le Geneva Business Center (environ 20 CHF pour la journée).

La conférence en chiffres :

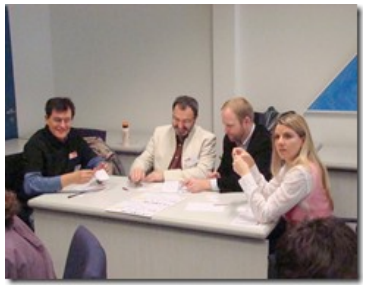
- deux "pistes parallèles" dans un auditorium et une grande salle
- douze sessions en tout, dont 3 en anglais
- une douzaine d'orateurs
- environ 95 personnes présentes (100 places maximum étaient possibles).

Comme a fallu faire des choix entre les sessions en parallèle, j'ai assisté à 6 d'entre elles, et parmi celles-ci, voici les 3 que j'ai préférées :

1. Apprenez les techniques de coaching avec le magicien d'Oz, par *Portia Tung* et *Pascal Van Cauvenberghe*.
2. Test automatiques autour d'un IHM, par *Didier Besset*.
3. Les 7 péchés capitaux du développeur, par *Freddy Mallet*.

J'étais très content de rencontrer enfin Portia et Pascal que je ne connaissais que virtuellement, ayant déjà utilisé avec succès certains de leurs jeux comme I'm not a Bottleneck! I'm a Free Man! (Lien71) J'ai bien apprécié la découverte d'un nouveau jeu, au contenu très utile puisqu'il consiste à travailler sur l'art de poser des questions et sur le co-coaching (ou peer coaching). Portia a tout d'abord captivé la salle avec un petit conte :

puis nous avons travaillé par groupes de 3 :



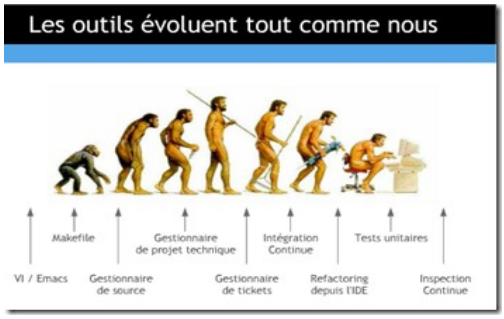
(ils sont 4, mais ce sont les autres orateurs, ils ont du avoir une dérogation). Voici une photo de l'ensemble de la salle :

Vous pouvez télécharger tout le support du jeu (intitulé The Yellow Brick Road) (Lien72). Et surtout, vous pouvez lire cette mise en application immédiate au retour de la conférence : Une rétrospective pas comme les autres (Lien73). Ce retour d'expérience tout frais montre bien l'intérêt d'une telle session dans une conférence, et plus généralement montre bien l'intérêt de nous enseigner régulièrement de nouvelles compétences non techniques (nous, les gens qui ont une formation technique). J'espère qu'il y a aura une session sur ce thème dans notre conférence agile à Grenoble en 2010. A suivre !

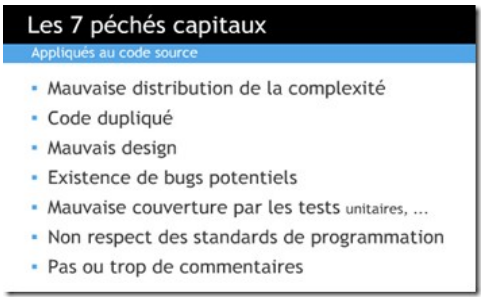
Le retour d'expérience sur les tests d'IHM m'a également beaucoup intéressé, et correspond bien à ce que j'attends d'une telle conférence : du concret, du factuel, des détails sur une application difficile mais compréhensible par tout le monde (contrôle du trafic aérien), du code, des idées à expérimenter au travail (la notion de primitives de tests), une piste théorique (les DSLs et les idées de Martin Fowler).

(Didier Besset qui présentait le retour d'expérience des tests IHM - et le nouveau T-shirt des organisateurs).

Enfin, Freddy Mallet nous a présenté SONAR (Lien74), un outil d'**inspection continue**



qui permet de chasser les "7 péchés capitaux" du développeur :

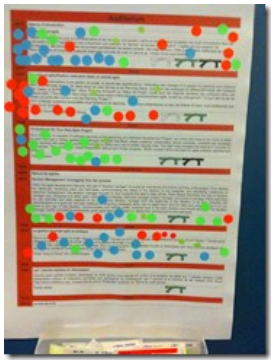


J'aurais préféré voir une plus longue démonstration "live" de l'outil, et moins de transparents, mais le sujet était intéressant et répond bien à mes préoccupations (j'ai fait pas mal d'expériences avec NDepend ces derniers temps). J'ai également apprécié le témoignage de l'orateur qui met en pratique ses idées, en continuant à toujours écrire des tests en premier pour le développement de son outil, et ce bien qu'il subisse une forte pression (il a monté sa propre entreprise). Hélas Freddy m'a confirmé que son outil ne permettait pas d'analyser le code Delphi, et c'est bien dommage pour nous qui avons une base de code importante dans ce langage.

J'ai moins accroché avec les autres présentations que j'ai suivies, car elles étaient plus éloignées de mes centres d'intérêt. Les 3 sessions ci-dessus correspondaient mieux à ce que j'attends de ce genre de conférences :

- sessions sur des outils non techniques (communication, coaching, développement de compétences)
- sessions sur des outils techniques permettant de réduire la dette technique, améliorer la qualité du code.
- retours d'expériences détaillés sur des logiciels de taille réaliste.

Voici les votes informels sur les sessions :



Vous devriez trouver prochainement les diverses présentations sur le site (Lien75). Certaines sont déjà disponibles sur les sites de leurs auteurs, par exemple celle de Thierry Cros est ici : XpDay Suisse : gouvernance agile (Lien76).
Dernière minute : le même Thierry vient de publier XpDay Suisse : impressions de voyage (Lien77).

Retrouvez ce billet sur le blog de Bruno Orsier : Lien78

Scrum et les changements pendant un sprint
Traiter les changements en flot continu avec une dose de Kanban

L'agilité se définit souvent comme la capacité à répondre aux changements, voire même à les favoriser. Par exemple, Scrum permet au client à travers son représentant (le Product Owner ou PO) d'incorporer des changements dans le périmètre fonctionnel à chaque fin de sprint. Pour beaucoup, c'est un progrès significatif. Maintenant Scrum se diffuse largement vers des équipes qui ne font pas que du développement de logiciel. Le paradoxe, c'est que nombreuses de ces équipes sont dans un mode de changements quasi-permanents, et le passage à Scrum peut représenter une transition rude en repoussant les changements au sprint suivant.
Cet article expose des pistes leur permettant une transition plus douce en introduisant une dose de Kanban dans un cadre Scrum.
Kanban est une pratique basée sur l'utilisation d'étiquettes (ou fiches) pour matérialiser des informations sur un processus, présentées sur un tableau, de façon similaire au tableau des tâches Scrum. Mais Kanban pousse à limiter le TAF (travail à finir), c'est-à-dire le nombre de travaux dans une étape du processus.

1. Prise en compte du changement dans les processus

Après le lancement de l'iPad durant le weekend du 3 avril, Apple n'a pas tardé à levé le voile sur la prochaine version de l'iPhone OS.

Mais avant de parler de l'iPhone OS proprement dit, Steve Jobs, aux commandes de cette keynote, est d'abord revenu sur le lancement de l'iPad.

1.1. Processus classique

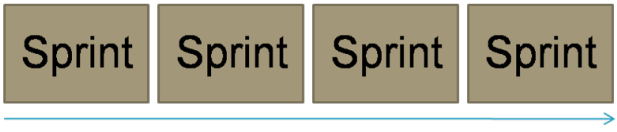
Quand le développement d'un nouveau projet s'effectue avec un cycle séquentiel (cycle en V ou en cascade), on évite le changement en essayant de le repousser à plus tard.

Pour cela, on essaie de définir le plus précisément possible ce qu'il faut faire en élaborant la spécification pour ensuite partir dans le développement sur cette base. Pendant le développement, les changements ne sont pas acceptés : ils sont repoussés dans la phase de maintenance.

Côté pratiques d'ingénierie, on sépare traditionnellement la gestion des exigences mise en oeuvre dès le début et la gestion des changements qu'on met en branle pour la phase de maintenance.

1.2. Processus agile

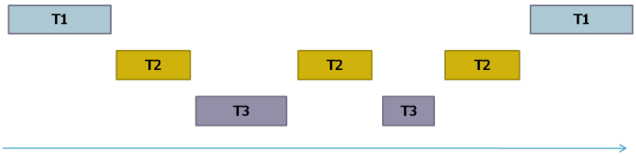
Avec un processus agile, le changement peut être pris en compte pendant le développement.



Typiquement avec Scrum, un développement est découpé en sprints. A chaque nouveau sprint, il est possible de prendre en compte et de traiter des changements qui sont intervenus pendant le sprint précédent.

1.3. Processus chaotique

Les changements arrivent fréquemment et sont traités de façon opportuniste en fonction de leur degré d'urgence. Bien souvent il n'y a pas de règle bien définie, les interruptions peuvent être nombreuses et provoquer des perturbations.



La tâche T1 est interrompue par la tâche T2 qui est elle-même interrompue par la tâche T3. Des perturbations incessantes ont un effet nocif sur la bonne santé des équipes. De plus les interruptions fréquentes sont génératrices de changements de contexte et donc de perte de temps.

1.4. Transition à Scrum

Scrum apporte une réponse à ces organisations. Pour la plupart des équipes, c'est une bonne nouvelle, à la fois pour celles qui n'acceptaient pas le changement et celles qui le pratiquaient de façon chaotique.

Cependant le passage à Scrum peut être considéré, en particulier pour ces dernières, comme trop difficile.

En effet, il existe des contextes où on ne peut pas supprimer complètement les perturbations pendant la durée du sprint. « Business first », le client sera toujours prioritaire dans des organisations qui ne peuvent pas, pour différentes raisons, dire non à une sollicitation d'un client, au moins dans la phase de transition à Scrum.

C'est ainsi qu'une équipe peut être perturbée par une démo à faire en urgence ou par un bug à corriger rapidement.

Dans ces cas là, l'approche préférable, inspirée du Kanban où la notion de flux contraint moins, permet d'accepter des changements sans attendre une fin de sprint comme avec Scrum.

Pas n'importe quand et n'importe comment cependant si on veut garder les avantages de Scrum.

Le cas, fréquent, où une seule équipe travaille sur plusieurs projets en même temps est aussi typique d'une équipe subissant des changements fréquents : le passage d'un projet à un autre s'apparente à une perturbation.

2. Retour sur la mécanique de Scrum

2.1. Scrum au pied de la lettre

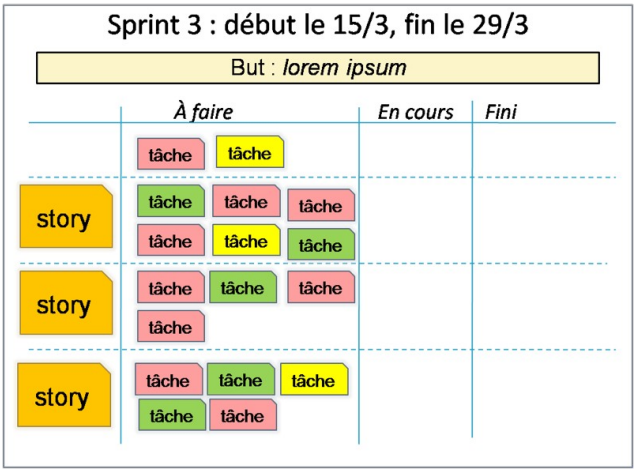
Pendant un sprint, l'équipe ne doit pas être perturbée, c'est une règle (au moins une recommandation de Scrum). Pas perturbée, cela signifie qu'on la laisse travailler pendant le sprint, selon l'objectif et le plan définis pendant la réunion de planification du sprint.

Concrètement, cela veut dire que personne n'a le droit d'ajouter du travail à la liste sur laquelle l'équipe s'est engagée sans son accord. Cela devrait aussi signifier que les ressources de l'équipe nesont pas changées (à la baisse) pendant le sprint ou encore que du travail ne constituant pas une story mais prenant du temps n'est pas ajouté non plus au tableau des tâches.

Diminuer les perturbations de cette façon a un effet positif. Cependant cela signifie que le traitement d'un changement est repoussé au sprint suivant. La durée des sprints peut être ajustée pour tenir compte de ce délai et réduire la gêne occasionnée.

2.2. Tâches associées à une story

C'est le tableau des tâches qui définit le travail à faire dans un sprint.



Dans un tableau Scrum, on trouve essentiellement des tâches associées aux stories. Elles représentent le travail détaillé à faire pour réaliser (et finir, selon la définition de ce que signifie fini) une story.

2.3. Tâches récurrentes

Il existe d'autres tâches, qui sont à faire pendant le sprint et qu'on ne peut pas associer à une story en particulier. Ces tâches, sans story associée, proviennent de la définition de fini pour un sprint. Par exemple, la mise à jour d'un document d'architecture, la production de rapports Sonar, le travail de test de charge, sont des tâches qu'on ne peut pas associer à une story spécifique.

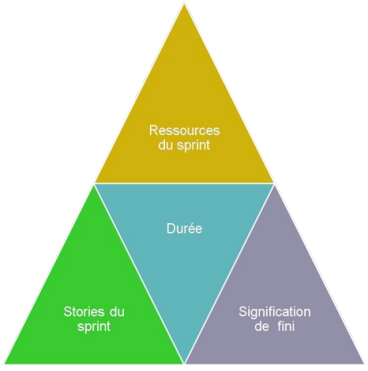
Ces tâches ont une autre particularité, en plus de ne pas être associée à une story, c'est qu'elles reviennent à chaque sprint (si la définition de fini ne change pas). On peut donc les qualifier de récurrentes. Elles sont relatives à la qualité du produit. Si elles ne sont pas faites, le sprint n'est pas prolongé (évidemment, cela serait contraire à la notion de bloc de temps), la vélocité n'est pas impactée, mais c'est la qualité du produit qui en pâtit. Cela produit de la dette technique si ces tâches ne sont pas faites, soit qu'on n'y a pas pensé et elles n'apparaissent pas dans la signification de fini, soit qu'elles y sont bien mais qu'on ne les a pas réalisées pendant le sprint.

2.4. Planification du sprint

Si on applique bien Scrum avec une signification de fini explicite, ces tâches récurrentes sont bien connues au début du sprint. Elles sont ajoutées dans le tableau des tâches au moment de la réunion de planification du sprint.

Si l'équipe travaille bien, elles seront finies à la fin du sprint et une autre instance de ces tâches seracréée lors du prochain sprint.

A la fin de la planification du sprint lorsque l'équipe s'engage, elle le fait en connaissant la durée du sprint, en connaissant les ressources (humaines) dont elle dispose, sur un périmètre défini par la liste des stories et en s'appuyant sur sa définition de ce que signifie fini.



En principe ces paramètres ne varient pas pendant le sprint. En pratique cela arrive tout de même. Pour rester dans le cadre Scrum, nous allons considérer que la durée du sprint, elle, ne peut pas changer pendant le sprint.

3. Changements pendant le sprint

On a vu que l'engagement de l'équipe au début d'un sprint était basé sur :

- Le périmètre fonctionnel de ce sprint, défini par la liste des stories prises en compte
- La signification partagée de fini, qui fait partie intégrante du « contrat » moral

- Les ressources dont disposait l'équipe

Un changement arrive quand une de ces grandeurs varie pendant le sprint. Examinons les différentes situations qui peuvent se présenter.

Éliminons tout de suite le cas où l'équipe identifie une nouvelle tâche nécessaire pour réaliser une story, qui avait été oubliée au début du sprint ou parce qu'une grosse tâche est décomposée. Il ne s'agit pas d'un changement puisque le périmètre fonctionnel n'est pas impacté. L'équipe a le droit de définir le travail à faire comme elle l'entend et donc d'ajouter une tâche liée à une story.

Les autres perturbations arrivant pendant le sprint et qui vont nécessiter du travail par l'équipe n'entrent pas dans ce cadre là, puisqu' elles vont revenir sur l'engagement pris en début du sprint.

3.1. Le périmètre du sprint varie

Si la demande de variation est à l'initiative de l'équipe, cela est un usage normal de Scrum. Une équipe qui se rend compte avant la fin du sprint qu'elle pourra en faire plus que prévu au début du sprint va demander une nouvelle story au Product Owner (ou l'inverse, d'ailleurs plus fréquent).

Si la demande est à l'initiative du Product Owner, c'est différent. Cela signifie qu'il a changé d'avis depuis la réunion de planification du sprint. Il s'est rendu compte qu'une story, qui était déjà dans le backlog de produit, est devenue plus prioritaire.

Scrum permet à une équipe de dire non. Elle peut donc refuser d'ajouter une nouvelle story demandée par le PO pendant le sprint. C'est déjà mieux que d'accepter sans conditions au risque de rater le sprint.

Une approche plus subtile est parfois possible, si la demande arrive alors que l'équipe n'a pas encore commencé à travailler sur des stories du sprint. Il est alors envisageable de proposer au PO d'échanger sa nouvelle story contre l'équivalent qui serait enlevé du périmètre du sprint. Ce troc de story est facilité par une estimation préalable en points.

3.2. La définition de fini varie

La signification de fini est partagée par toute l'équipe. En principe elle est définie au début du projet et peut varier légèrement à chaque sprint. Le meilleur moment pour discuter de sa mise à jour est pendant la rétrospective de sprint.

Il peut arriver que l'équipe décèle le besoin de revoir la signification de fini pendant le sprint et que cette révision entraîne du travail à faire.

Quelques exemples : la production ou la mise à jour d'un document transverse, comme celui d'architecture, la mise en place d'un script pour automatiser le déploiement ...

C'est à l'équipe de décider, par exemple lors du scrum quotidien, si cette révision peut attendre le sprint suivant ou doit être entreprise au plus vite. Dans ce dernier cas, le travail à faire est considéré comme une tâche urgente, notion sur laquelle nous allons revenir.

3.3. Les ressources du sprint varient

Ce cas se produit quand du travail est demandé à un ou plusieurs membres de l'équipe, alors qu'il n'était pas prévu au début du sprint et qu'il ne s'agit pas d'une story déjà identifiée dans le backlog de produit.

Le travail demandé peut être en relation avec le produit que l'équipe développe. Cela arrive fréquemment quand l'équipe n'est pas confinée au développement, mais s'occupe aussi du support et de l'avant-vente.

Voici quelques exemples de ce qui peut arriver :

- Un défaut (bug) a été trouvé sur le produit, qu'il soit en production (ce qui fait augmenter le degré d'urgence) ou pas
- Du feedback est remonté du PO ou d'utilisateurs privilégiés utilisant le résultat du sprint précédent
- Une démo doit être faite très vite pour un (futur) client important
- Une question posée sur le forum des utilisateurs nécessite une réponse rapide

Parfois une ressource peut être demandée pour travailler sur quelque chose qui n'a rien à voir avec le produit développé.

Les postures possibles pour une équipe Scrum sont les suivantes :

- Dire non. Il est probable que certains des travaux demandés peuvent attendre le sprint suivant. La solution est alors pour les demandeurs de ces travaux (en général le PO) d'ajouter une entrée dans le backlog de produit. Mais ce n'est pas possible pour tous les travaux, certains ayant un véritable caractère d'urgence.
- Dire oui. Le problème est que cela va avoir un impact sur l'engagement de l'équipe du début de sprint, puisqu'elle va passer du temps sur ces tâches non prévues. Si cela n'est pas contrôlé on en revient au chaos des changements permanents. Un des risques est que ces travaux n'apparaissent pas explicitement dans le tableau des tâches.

Pour conserver deux notions essentielles de Scrum, la transparence et le timebox du sprint, il est préférable que ces travaux apparaissent explicitement dans le tableau des tâches.

L'approche recommandée est donc d'identifier les tâches vraiment urgentes et de les traiter de façon spécifique, à l'intérieur du sprint Scrum. Les événements qui ne seront pas considérés comme urgents et donc pas traités dans le sprint courant, il convient d' en faire une story qui va dans le backlog de produit.

3.4. Un événement prévisible n'est pas vraiment une perturbation

Dans certaines circonstances, on peut parler de perturbation prévue à l'avance. C'est le cas lorsqu'un événement ponctuel a lieu pendant le sprint. Cela peut-être

la participation à un salon, ou une personne qui part en formation. Une perturbation c'est du travail en plus ou une ressource en moins. Si l'équipe dispose d'un calendrier visuel, l'événement y aura naturellement sa place.

A partir du moment où c'est prévu au début du sprint, cela peut être pris en compte pendant la réunion de planification. L'équipe s'engage en connaissance de cause et on ne peut plus parler de perturbation.

3.5. Des obstacles sont rencontrés

Un autre cas de tâche urgente concerne le travail à faire pour l'élimination d'un obstacle rencontré pendant le sprint.

Exemple : un serveur qui tombe en panne.

4. Traitement des tâches urgentes : vers du Kanban

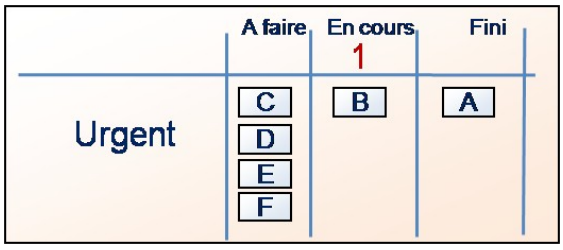
Une tâche urgente arrive, la première personne qui finit sa tâche en cours la prend, avant d'en commencer une autre.

Une façon plus proactive de procéder avec les tâches urgentes, est de limiter leur nombre par sprint en fixant et ajustant une limite, le TAF. Cette limite peut être globale, par état ou par personne de l'équipe.

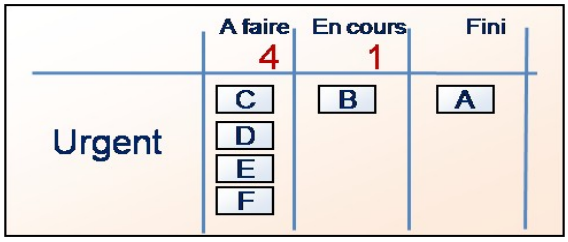
Nous sommes là plus proches du Kanban que du Scrum pur. En particulier si on considère qu'il faut limiter le nombre de tâches perturbantes à un moment donné. La limitation du nombre de travaux dans un état constitue un élément fondamental du Kanban, la limite est appelée TAF (WIP en anglais, c'est-à-dire le travail à faire dans le processus, qu'il faut limiter).

4.1. Limite par état

On suppose qu'une tâche urgente peut être dans un des trois états suivants : à faire, en cours et finie. Le plus usuel est de limiter le nombre de tâches urgentes en cours. Une limite de 1 oblige à avoir fini une tâche urgente avant d'en traiter une nouvelle, ce qui est une bonne chose.

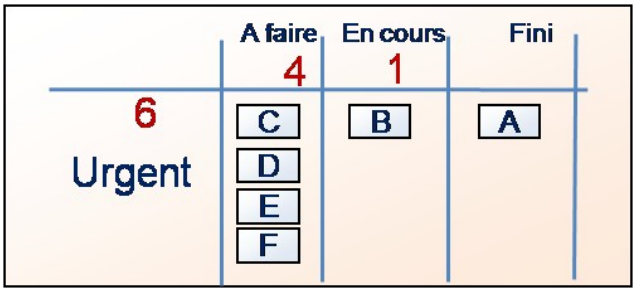


On peut aussi limiter le nombre de tâches dans l'état à faire. Cela contraint le PO à définir les priorités ou le degré d'urgence. Si le TAF dans l'état à faire est limité à 4 et que les 4 places sont prises, le PO peut introduire une nouvelle tâche urgente, mais à condition d'en enlever une.



4.2. Limite globale

Le nombre de tâches urgentes est limité dans un sprint. Quel que soit leur état. Le PO ne peut plus en ajouter lorsque la limite est atteinte. La limite se discute à chaque rétrospective.



4.3. Limite par personne

Si la limite de TAF dans l'état en cours est supérieure à 1, il peut être intéressant d'empêcher qu'une seule personne ait plusieurs tâches urgentes en cours.

4.4. Usage du mou

Le mou permet de faire des tâches non prévues sans remettre en cause les objectifs du sprint.

Dans la boite de temps que constitue le sprint, on peut considérer que lors de la réunion de planification du sprint, une partie est allouée à la réalisation des stories, une autre aux tâches récurrentes pour satisfaire la signification de fini et qu'il faut garder du mou pour les incertitudes sur les estimations et pour les tâches urgentes.

En lissant le temps passé sur les tâches urgentes (avec une des techniques de limitation de TAF), cela revient à garder dans chaque sprint une partie du temps pour traiter l'urgence.

4.5. Impact sur la vélocité

Les tâches n'ayant pas de points associés, à la différence des stories, elles ne contribuent pas à la vélocité. Mais

comme on y passe du temps, ce temps n'est pas passé sur des stories et donc l'impact indirect est de diminuer la vélocité.

Les tâches récurrentes prennent à peu près le même temps à chaque sprint. Leur impact sur la vélocité sera donc à peu près le même pour chaque sprint. Avec l'usage fait de la vélocité, cela revient à dire que les tâches récurrentes n'ont aucune conséquence.

Les tâches urgentes peuvent varier d'un sprint à l'autre mais la variation est limitée par le TAF.

5. En résumé

Pendant un sprint, il faut s'efforcer de limiter les perturbations. Cela peut se faire de façon proactive lors de la planification du sprint en identifiant comme des stories les événements perturbateurs et de façon réactive en repoussant un changement au sprint suivant.

Si malgré tout, il y a une tâche urgente ajoutée pendant le sprint, cette tâche doit être prise et commencée dès qu'une ressource finit son travail en cours.

Il est préférable de limiter le nombre de tâches urgentes pendant un sprint. La limite peut porter sur le nombre de tâches traitées pendant le sprint ou sur le nombre de celles qui peuvent traitées de façon simultanée ou sur celles que peut faire une personne.

Seule l'équipe (y compris le PO bien entendu) a le droit d'ajouter une tâche urgente pendant le sprint.

Exemples

1. Vous êtes dans un sprint. Les stories S1, S2 et S3 sont dans l'objectif du sprint
2. Le PO veut ajouter S4. Si S3 n'est pas commencé et a la même taille on peut substituer S3 par S4. Si tout est commencé ou qu'il ne reste qu'une story de taille inférieure, on refuse.
3. Le boss a besoin de quelqu'un de l'équipe pour préparer une démo à un client important. On lui répond qu'il aurait pu la prévoir et on lui demande s'il peut la décaler au sprint suivant. S'il faut absolument la faire, c'est une tâche urgente qui suit les règles de TAF définies par l'équipe.

6. Références

Kanban et Scrum, tirer le meilleur des deux (Lien79) par Henrik Kniberg et Mattias Skarin, traduit en français par Claude Aubry, Frédéric Faure, Antoine Vernois et Fabrice Aimetti.

Retrouvez l'article de Claude Aubry en ligne : Lien80

Liens

Lien1 : http://www.eclipse.org/articles/StyledText%201/article1.html

Lien2 : http://www.eclipse.org/swt/
Lien3 : http://wiki.eclipse.org/index.php/Rich_Client_Platform
Lien4 : http://www.eclipse.org/articles/Article-Forms/article.html
Lien5 : http://david-chautard.developpez.com/tutoriels/java/raccourcis-clavier-styledtext/
Lien6 : http://www.google.com/codesearch/p?hl=en&sa=N&cd=3&ct=rc#uX1GffpyOZk/core/java/android/provider/Calendar.java
Lien7 : http://developer.android.com/reference/android/provider/ContactsContract.html
Lien8 : http://developer.android.com/resources/samples/ContactManager/index.html
Lien9 : http://android.developpez.com/faq/
Lien10 : http://developer.android.com/reference/java/lang/Runnable.html
Lien11 : http://developer.android.com/reference/java/lang/Thread.html
Lien12 : http://davy-leggieri.developpez.com/tutoriels/android/threads-composants-application/
Lien13 : http://x-plode.developpez.com/tutoriels/netbeans/logiciel-test-data-warehouse-sur-plateforme-netbeans/
Lien14 : http://www.developpez.net/forums/d825811/php/langage/developpement-php6-suspendu-reprendra-t/
Lien15 : http://www.developpez.net/forums/d879824/php/bibliotheques-frameworks/symfony/sensio-labs-presente-symfony-2-a/
Lien16 : http://www.drupalgardens.com/
Lien17 : http://www.developpez.net/forums/d893752/club-professionnels-informatique/actualites/cms-open-source-drupal-sortira-version-hebergee-courant-2010-a/
Lien18 : http://dico.developpez.com/html/1096-Gestion-de-projet-cycle-de-vie.php
Lien19 : http://phing.info/docs/guide/stable/
Lien20 : http://phing.info/trac/wiki/Users/Documentation/CruiseControl
Lien21 : http://pear.php.net/manual/fr/installation.getting.php
Lien22 : http://phing.info/trac/wiki/Users/Download
Lien23 : http://dico.developpez.com/html/1746-Generalites-PATH.php
Lien24 : http://farcellier.developpez.com/tutoriels/php/phing-gerer-cycle-vie-projet/fichiers/package_exemples.zip
Lien25 : http://farcellier.developpez.com/tutoriels/php/phing-gerer-cycle-vie-projet/
Lien26 : http://www.youtube.com/watch?v=fyfu4OwjUEI
Lien27 : http://code.google.com/p/quake2-gwt-port/
Lien28 : http://code.google.com/p/quake2-gwt-port/wiki/BuildingAndRunning
Lien29 : http://blog.developpez.com/ddelbecq/p8786/java/quake-2-en-javascript-grace-a-html5/
Lien30 : http://www.developpez.net/forums/d838682/club-professionnels-informatique/actualites/ie9-jouira-dexcellentes-capacites-dacceleration-supportera-nouveaux-standards/
Lien31 : http://www.google.com/ads/preferences
Lien32 : http://www.developpez.net/forums/d899839/club-professionnels-informatique/actualites/google-lance-service-remarketing-permet-suivre-l-internaute-partout-travers-web/
Lien33 : http://www.ibordeaux.fr/
Lien34 : http://www.irennes.fr/
Lien35 : http://www.wimt.fr/
Lien36 : http://www.street-invaders.net/
Lien37 : http://www.developpez.net/forums/d906432/club-professionnels-informatique/actualites/gagnants-challenge-mappy-veulent-rendre-api-open-source-interview-exclusive-developpeurs/
Lien38 : http://www.developpez.net/forums/d902045/club-professionnels-informatique/actualites/attaques-web-2-0-ont-augmente-500-tour-dhorizon-dernieres-tendances-cybercriminelles/
Lien39 : http://ritter-jack.developpez.com/tutoriels/javascript/realiser-slider-facilement-grace-plug-in-jquery-coda-slider-v-2/exemple/slider.html
Lien40 : ftp://ftp-developpez.com/ritter-jack/tutoriels/coda-slider/Exemple.rar
Lien41 : http://www.ndoherty.biz/
Lien42 : http://www.ndoherty.biz/demos/coda-slider/2.0/
Lien43 : http://www.ndoherty.biz/forums/viewtopic.php?f=4&t=2
Lien44 : http://ritter-jack.developpez.com/tutoriels/javascript/realiser-slider-facilement-grace-plug-in-jquery-coda-slider-v-2/
Lien45 : http://www.mpprogramming.com/
Lien46 : http://come-david.developpez.com/tutoriels/castor/
Lien47 : http://herbsutter.wordpress.com/2010/03/13/trip-report-march-2010-iso-c-standards-meeting/
Lien48 : http://www.developpez.net/forums/d891380/c-cpp/cpp/cpp0x-draft-final-ete-vote/
Lien49 : http://www.developpez.net/forums/d885572/c-cpp/cpp/penser-frameworks-architectures-font-deriver-toutes-classes-superobjet/
Lien50 : http://doc.qt.nokia.com/4.7-snapshot/qt4-7-intro.html
Lien51 : http://www.kyte.tv/ch/qtorials/handling-multiple-targets-in-qt-cre/i=601&s=828265
Lien52 : http://www.developpez.com/redirect/60
Lien53 : http://www.developpez.net/forums/d811427/c-cpp/bibliotheques/qt/qt-quick-futur-developpement-dihm-anciennement-declarative-ui/

Lien54 : http://www.developpez.net/forums/d844607/c-cpp/bibliotheques/qt/sortie-qt-4-6-2-a/
Lien55 : http://www.developpez.net/forums/d825667/c-cpp/bibliotheques/qt/qt-4-7-0-tech-preview-qt-creator-2-0-alpha/#post5059577
Lien56 : http://www.developpez.net/forums/private.php?do=newpm&u=292096
Lien57 : http://cpp.developpez.com/cours/stl/
Lien58 : http://doc.trolltech.com/4.6-snapshot/templates.html
Lien59 : http://lists.trolltech.com/qt-interest/2002-08/thread00000-0.html
Lien60 : http://qt.developpez.com/faq/?page=generalites-compil
Lien61 : http://doc.trolltech.com/4.6-snapshot/moc.html
Lien62 : http://qt.nokia.com/doc/latest/properties.html
Lien63 : http://qt.nokia.com/doc/latest/qmetaproperty.html
Lien64 : http://louis-du-verdier.developpez.com/qt/fondations/
Lien65 : http://mac.developpez.com/livres/
Lien66 : http://mac.developpez.com/evenements/2010/iPhoneOS4/
Lien67 : http://www.xpday.ch/
Lien68 : http://conception.developpez.com/reportage/agilite/xp-day-suisse-2009/
Lien69 : http://www.developpez.com/
Lien70 : http://agile-alchemist.com/
Lien71 : http://www.agilecoach.net/coach-tools/bottleneck-game/

Lien72 : http://www.agilefairytales.com/dist/The-Yellow-Brick-Road-2-0.zip
Lien73 : http://luc-jeanniard.blogspot.com/2010/03/une-retrospective-pas-comme-les-autres.html
Lien74 : http://sonar.codehaus.org/
Lien75 : http://www.xpday.ch/
Lien76 : http://etreagile.thierrycros.net/home/index.php?post/2010/03/31/XpDay-Suisse-%3A-gouvernance-agile
Lien77 : http://etreagile.thierrycros.net/home/index.php?post/2010/03/31/XpDay-Suisse-%3A-impressions-de-voyage
Lien78 : http://blog.developpez.com/bruno-orsier/p8781/developpement-agile/compte-rendu-xp-day-suisse-2010/
Lien79 : http://www.aubryconseil.com/post/2eme-version-francaise-de-Kanban-et-Scrum
Lien80 : http://claude-aubry.developpez.com/articles/scrum/kanban/gestion-changement/