N°167 JANVIER 2014

GNU LINUX MAGAZINE / FRANCE

WEBDESIGN / FLASK

Créez une interface Web « clicka-convi » sans sombrer dans la folie : exemple avec Flask, WTForms et Bootstrap p.08

ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

METEORJS / NOCODE

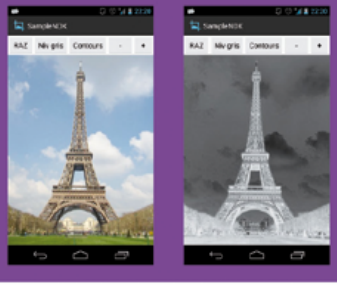
Concrétisez vos aspirations de développeur web avec un framework qui n'est pas fait pour les développeurs web p.12

PHP / LOGS

Adoptez enfin une méthodologie rigoureuse pour les logs de votre application web PHP avec Apache log4php p.58

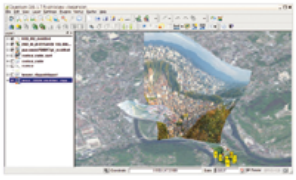
ANDROID / C++

Réutilisez du code C/C++ natif sous Android avec le Native Development Kit : exemple de traitement d'images p.78



L 19275 - 167 - F: 7,90 € - RD

SCIENCE / IMAGES



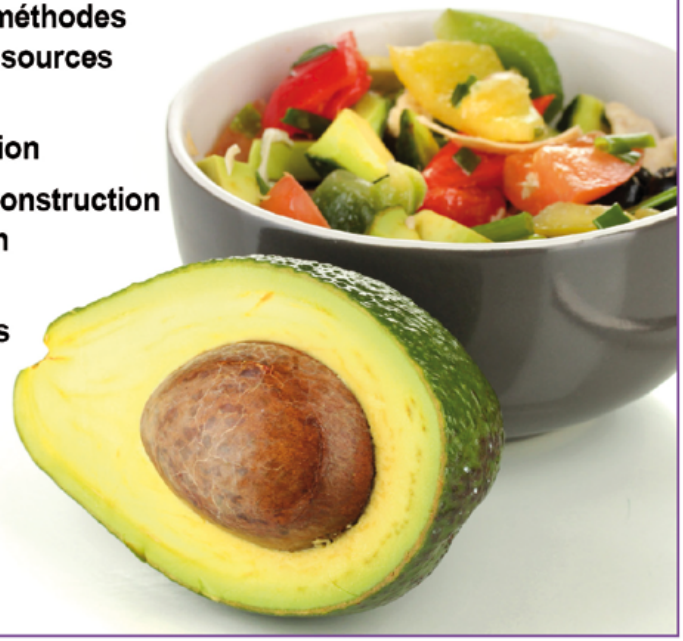
Correction géométrique et projection sur modèle numérique d'élévation pour exploitation quantitative de photographies numériques p.42

NOYAU / SYSADMIN

Qui a dit qu'il n'était plus nécessaire de s'en occuper ?

ADMINISTREZ ET CONFIGUREZ LE KERNEL p.24

1. Les différentes méthodes pour obtenir les sources
2. Configuration et personnalisation
3. Compilation et construction standard, Debian et Fedora
4. Installation, tests et boot
5. Utilisation de DKMS pour la gestion de modules



BUS / LIGHT

Découvrez Abus, un bus logiciel à faible empreinte mémoire, sans démon central, disposant de RPC et d'une API simple p.63

PHP / NOUVAUTES

PHP 5.3 est arrivé en fin de vie ! Il est vraiment temps de migrer vers PHP 5.5 et ses dernières nouveautés p.04

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD

Le CNLL
Conseil National du Logiciel Libre
et ses représentants régionaux
PRÉSENTENT

les RENCONTRES REGIONALES du LOGICIEL LIBRE & du SECTEUR PUBLIC

Du 04 Octobre 2013 au 17 Avril 2014

Paris
Lille
Metz
Strasbourg
Lyon
Rennes
Brest
Nantes
Bordeaux
Toulouse

Venez rencontrer vos pairs
avec des problématiques communes !

Venez rencontrer
les prestataires locaux de l'OpenSource !

En savoir plus sur **www.rrll.fr**

Evénement soutenu par
ADOSIS, *Agence Digitale*, spécialiste OpenSource
www.adosis.com

SOMMAIRE N°167

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax :** 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – boutique.ed-diamond.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Sittler
Réalisation graphique : Jérémy Gall

Responsable publicité : Valérie Fréchard, Tél. : 03 67 10 00 27
v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne

Distribution France : (uniquement pour les dépositaires de presse)

MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04

IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, N° **ISSN :** 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,90 €

MIXTE Papier issu de sources responsables FSC® C015136 www.fsc.org

Membre April www.april.org

ÉDITORIAL

« Bah ! Je n'ai rien à me reprocher moi »

Voici généralement l'argument largement entendu de ceux qui n'ont pas d'avis ou sont potentiellement favorables à la surveillance globale et systématique : « *De toute façon, on nous surveille déjà et je m'en fiche parce que, moi, je ne fais rien de mal, je n'ai rien à cacher* ».

Ceci peut sembler être un argument imparable si on retire purement et simplement l'aspect moral de la question. Pourtant, c'est un raisonnement totalement faux. En effet, dans la société telle qu'elle existe, ce n'est pas un individu qui détermine ce qui est bien ou mal, juste ou injuste, normal ou déviant, entendu ou inacceptable... Ce sont les normes dictées par la société elle-même. Ces notions sont donc parfaitement hors de contrôle pour l'individu isolé, variables géographiquement et promptes à changer dans le temps.

Ainsi, l'argument qui consiste à dire qu'on n'a rien à se reprocher ne peut être valable qu'accompagné des mots « maintenant », « ici » et « selon moi-même ». En d'autres termes, ce n'est pas vous qui décidez si vous faites quelque chose de mal ou non, mais la société, selon les normes et les lois établies par et pour la société (en principe).

Mais la surveillance globale, au cœur de toutes les inquiétudes actuellement, implique d'autres risques, découlant de l'interprétation des informations. Lorsqu'on peut collecter toutes les données que l'on souhaite sur « une cible », on peut littéralement faire dire tout ce que l'on souhaite à ces mêmes données. Pire encore, si l'analyse est faite par un ensemble d'algorithmes, le risque de faux positifs est bel et bien présent. Même s'il ne s'agit que d'un faible pourcentage, classer automatiquement une personne dans les « individus à risque » ou les « possibles terroristes » n'a rien à voir avec le fait de placer un fichier en quarantaine ou un mail dans les « spams probables ».

À l'heure où je rédige cet édito, le Conseil Constitutionnel n'a pas été saisi concernant la loi de programmation militaire (LPM) et son fameux article 13 (devenu 20). Lorsque vous lirez ceci, il est donc possible que cette loi soit appliquée (décrets d'application) et si tel est le cas, je pense qu'il peut être tout à fait raisonnable de commencer à réviser vos classiques, en commençant par Orwell...

Bonne année 1984^W2014 !

Denis Bodor

LES NOUVEAUTÉS DE PHP 5.5

par Stéphane Mourey [Taohacker]

Il y a quelques mois, la version 5.5 de PHP est sortie de son état de Release Candidate pour devenir la dernière version officielle. Moins attendue sans doute que la version 5.4 (qui offrait enfin une solution au problème de l'héritage multiple), la nouvelle version n'en offre pas moins quelques nouveautés intéressantes que nous allons découvrir dans cet article.

1 Les générateurs

Depuis la version 5, PHP fournit une interface **Iterator** qui permet de construire des itérateurs. Les itérateurs sont des objets qui permettent d'accéder à un ensemble de données, d'objets ou de ce que vous voulez, pour pouvoir le parcourir comme un tableau. Cette possibilité, très intéressante et très puissante, est toutefois un peu complexe à implémenter : il y a cinq méthodes à écrire pour remplir les exigences de l'interface, à savoir **current**, **key**, **next**, **rewind** et **valid**. Ces méthodes permettent de s'assurer que l'itérateur se comportera comme un tableau dans les différentes situations où il aura à le faire.

Il se peut toutefois que, tout en voulant parcourir votre ensemble comme un tableau, vous n'ayez pas besoin de toute la souplesse que cela implique. Si vous avez simplement besoin d'avancer dans votre ensemble, sans avoir jamais besoin de le réinitialiser ou d'aller en arrière, vous pourrez utiliser un générateur. Celui-ci est beaucoup plus simple à mettre en œuvre. En effet, pour créer un générateur, vous n'avez besoin d'écrire qu'une seule fonction...

Dans cette fonction, vous allez parcourir votre ensemble et à chaque fois que vous disposerez d'un élément de votre ensemble sur lequel vous voulez travailler, vous le renverrez à l'aide de l'instruction **yield** au lieu de **return**. La différence entre **yield** et **return** est que **yield** ne termine pas le fonctionnement de la fonction mais le suspend. La fonction reprend exactement là où elle en était lorsqu'on lui demande un nouvel élément de l'ensemble.

Imaginons par exemple que vous vouliez parcourir les résultats d'une fonction sur un intervalle donné d'entiers. Comme peu importe la fonction dans notre exemple, nous l'appellerons **fonction()**. Nous pouvons donc construire un générateur de la façon suivante :

```
function gererateurPourFonction($debut,$fin) {
  for ($i=$debut;$i<=$fin;$i++) {
    yield fonction($i);
  }
}
```

Ensuite, nous pouvons utiliser ce générateur dans une boucle **foreach**, éventuellement après lui avoir donné un nom de variable pour la lisibilité :

```
$monGenerateur = generateurPourFonction(1,10);
foreach ($monGenerateur as $valeur) {
  echo $valeur."\n";
}
```

Si vous souhaitez que votre générateur retourne une paire clé / valeur à chaque itération, et vous permette du coup de construire un tableau associatif, vous utiliserez l'instruction yield en utilisant la syntaxe suivante :

```
yield $clef => $valeur;
```

Si vous souhaitez renvoyer une valeur nulle, utilisez yield sans paramètre :

```
yield;
```

Enfin, il est possible que votre générateur renvoie ses valeurs par référence, ce qui est intéressant pour limiter l'empreinte mémoire de votre script. Pour ce faire, vous procédez comme pour les autres fonctions, en précédant leur nom du signe esperluette « & », tant dans sa déclaration que dans son utilisation :

```
function &gererateurIntervalleDEntiers($debut,$fin) {
  for ($i=$debut;$i<=$fin;$i++) {
    yield $i;
  }
}
foreach (&gererateurIntervalleDEntiers(1,10) as
$valeur) {
  echo $valeur."\n";
}
```

Enfin, pour conclure sur les générateurs, si réellement vous avez besoin de revenir en arrière dans votre parcours, la seule solution que vous avez est de cloner votre objet générateur à l'aide de l'instruction **clone**. Vous obtiendrez alors un générateur identique, mais réinitialisé : vous pourrez à nouveau le parcourir, mais en partant du début.

2 OPcache

Déjà disponible depuis la version 5.2, l'extension OPcache est maintenant présente par défaut dans PHP 5.5. Cette extension vise à stocker en cache le bycode des scripts précompilés dans la mémoire partagée, de sorte qu'il n'est plus nécessaire à PHP de charger et d'analyser les scripts à chaque demande. Il en résulte un gain de performance non négligeable.

Si vous utilisez l'extension Xdebug conjointement à celle-ci, il vous faudra prendre garde à charger OPcache en premier.

Pour bénéficier de cette extension, il n'y a rien à faire de particulier au niveau du code, tout se fait au niveau du fichier **php.ini**. Il faut ajouter (ou dé-commenter) les lignes suivantes :

```
[opcache]
opcache.enable=1
```

D'autres directives ont été rajoutées dans la configuration, et, curieusement, certaines des valeurs par défaut (du moins avec ma distribution Arm Arch Linux) ne correspondent pas à celles recommandées dans la documentation. J'ai résumé les différences dans le tableau ci-dessous :

Directive	Valeur par défaut	Valeur recommandée
opcache.memory_consumption	64	128
opcache.interned_strings_buffer	4	8
opcache.max_accelerated_files	2000	4000
opcache.revalidate_freq	2	60
opcache.fast_shutdown	0	1
opcache.enable_cli	0	1

Personnellement, je diverge de la configuration recommandée au moins sur **opcache.enable_cli** : j'utilise en général PHP en ligne de commande avec divers outils de développement (PHPUnit, PHPSniffer...) ou pour des tests, beaucoup plus rarement pour réaliser des scripts que j'utiliserai ensuite (bash est quand même plus adapté pour ça, vous ne croyez pas ?). Du coup, il n'y a pas d'intérêt à conserver ce code en cache, puisqu'il sera rarement exécuté à l'identique.

3 finally

Une nouvelle instruction apparaît, permettant d'améliorer le traitement des exceptions, il s'agit de **finally**. Cette instruction ouvre un bloc de code après les blocs **try** et **catch**. Ce bloc **finally** sera toujours exécuté qu'une exception ait été levée ou pas, y compris si une instruction **return** se produit dans les blocs **try** et **catch**. Cela permet de garantir que le code sera toujours exécuté, quel que soit le résultat de l'essai. Il est même possible d'y intercepter l'exception pour en lever une autre...

```
function test() {
  try {
    $mysqli = new mysqli('localhost', 'my_user', 'my_password', 'my_db');
    if ($mysqli->connect_error) throw new \Exception("Impossible de connecter !");
  } catch (\Exception $e) {
    echo $e->getMessage()."\n";
  } finally {
    echo "Ce code sera toujours exécuté!";
    throw $e;
  }
}
```

L'intérêt de **finally** est principalement de s'assurer que la levée d'une exception n'empêche pas une remise en ordre nécessaire avant une éventuelle interruption du traitement en cours.

4 Nouvelle API de hachage de mots de passe

Bien souvent, on reproche à PHP ses faiblesses du côté de la sécurité. Et il faut bien avouer ce qui est, PHP réserve quelques mauvaises surprises de ce côté-là. Mais plus grave encore que la conception du langage, le fait est que beaucoup de programmeurs PHP écrivent leur code sans avoir même connaissance des bonnes pratiques en ce domaine. Ce n'est que dans la douleur et suite à de nombreuses attaques que la communauté des développeurs PHP a pris conscience de l'importance de l'enjeu, même si beaucoup de sonnettes d'alarme avaient été tirées depuis longtemps.

L'un de ces faiblesses était le stockage des mots des passe utilisateurs. Trop souvent stockés en clair, simplement hachés grâce à une fonction md5 défectueuse dans certains cas, de mauvaises pratiques de cryptage, il y a bien des raisons expliquant les faiblesses des applications PHP. Avec la version 5.5, l'équipe de PHP tente de résoudre le problème en proposant une nouvelle API de hachage de mots de passe qui prend en charge toute la complexité nécessaire à la sécurité requise pour une telle tâche, et ce, de façon transparente pour le développeur. Pour ceux qui seraient sensibles à la question mais qui n'auraient pas la possibilité de migrer tout de suite, sachez qu'au départ cette API a été implémentée en PHP (version supérieure ou égale à 5.3.7) et peut fonctionner dans l'espace utilisateur. Vous trouverez les sources de cette dernière à cette adresse : https://github.com/ircmaxell/password_compat. Les deux versions sont compatibles.

Mais avant d'étudier les nouvelles fonctions proposées par cette API, laissez-moi vous rappeler un point de la législation française qu'il est important de connaître si vous offrez un service en ligne accessible au grand public. Le Décret n° 2011-219 du 25 février 2011 relatif à la conservation et à la communication des données permettant d'identifier toute personne ayant contribué à la création d'un contenu mis en ligne (consultable ici : http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000023646013&dateTexte=20130822), décret d'application de la fameuse Loi sur la Confiance en l'Économie Numérique (LCEN, http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000801164&dateTexte=20130822), mentionnait l'obligation pour les personnes offrant

un service en ligne de conserver le mot de passe des utilisateurs en clair. Ce qui était, du point de vue de la sécurité informatique, une véritable aberration : un administrateur (sauf s'il travaille avec Windows;)) n'a pas besoin de connaître les mots de passe de ses utilisateurs et des mots de passe correctement cryptés garantissent que si la base de données qui les contient a été compromise, ceux-ci ne pourront pas être utilisés facilement. Nombre de prestataires ont refusé de se plier à la loi sur ce point et certains, dont Google et Facebook ont fait pression sur le Gouvernement pour la modifier. Il semble qu'ils aient eu gain de cause, car le 1er avril 2012, le décret a été modifié (http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000023646013&dateTexte=20130822). Il n'est maintenant plus question que de conserver « les données permettant de vérifier le mot de passe ou de le modifier, dans leur dernière version mise à jour ». Autrement dit, uniquement des informations que nous avons de toute façon besoin de connaître. Nous pouvons donc utiliser cette nouvelle API sereinement, ce qui n'était pas le cas il y a quelques mois seulement...

Passons maintenant à l'aspect concret. La première chose à savoir sur cette nouvelle API est que l'algorithme utilisé pour effectuer le hachage recommandé est déterminé par une nouvelle constante prédéfinie, à savoir **PASSWORD_DEFAULT**, dont la valeur est aujourd'hui la même que celle d'une autre constante prédéfinie, à savoir **PASSWORD_BCRYPT**, indiquant que l'algorithme **CRYPT_BLOWFISH** sera utilisé par les fonctions de hachage. Aujourd'hui, il est le seul algorithme supporté, et on pourrait être tenté de négliger cet aspect de la question.

De nouveaux algorithmes, supposés plus performants, sont amenés à être ajoutés et l'un d'entre eux pourrait un jour remplacer **CRYPT_BLOWFISH** comme algorithme par défaut dans une version future de PHP. Le jour où cela se produira, la fonction de hachage fournira des résultats différents par défaut. Dès lors, vous pourriez craindre que vos utilisateurs ne puissent plus s'identifier. Cette crainte, bien que légitime, n'est pas fondée. En effet, la signature produite par la fonction de hachage contient l'indication de l'algorithme utilisé, de même que les options avec lesquelles il a été utilisé (grain de sel, coût). Vous pouvez donc à tout moment changer l'algorithme de cryptage, de même que ces options, sans craindre une incompatibilité avec les anciennes signatures. Personnellement, bien que je n'ai pas suivi les débats qui ont menés à ce choix, et bien que je n'aie pas analysé comment ces informations sont stockées dans la signature, j'ai le sentiment que l'intérêt du grain de sel et du changement d'algorithme est grandement limité : si PHP est en mesure d'y retrouver ces informations, un attaquant le sera également... j'aurai tendance à vouloir rajouter mon grain de sel par dessus celui de PHP ! Je me contenterai toutefois ici de montrer comment la documentation recommande d'utiliser cette API.

Pour hacher un mot de passe, il faut utiliser la fonction **password_hash**. Celle-ci accepte trois arguments : le premier, obligatoire, est le mot de passe à hacher ; le second, obligatoire, est une constante prédéfinie indiquant l'algorithme à utiliser, **PASSWORD_DEFAULT** est recommandé par défaut; le dernier est un tableau contenant des arguments nommés destinés à l'algorithme de cryptage (ce fonctionnement permet à la fonction de conserver la même signature quel que soit l'algorithme à utiliser).

En principe, vous n'avez pas à vous occuper des arguments optionnels, le fonctionnement par défaut étant satisfaisant. Pour informations, sachez qu'en utilisant l'algorithme **CRYPT_BLOWFISH**, le tableau d'options peut en contenir deux, à savoir **salt**, le grain de sel qui vient rendre le hachage plus compliqué à casser, et **cost**, qui définit le nombre d'itérations effectuées par l'algorithme, et, par conséquent son coût en terme de ressources. Si vous ne renseignez pas ces options, un grain de sel aléatoire est utilisé pour chaque hachage et 10 est le **cost** par défaut.

Voici donc la ligne de code recommandée pour effectuer un hachage de mot de passe :

```
$hash = password_hash($password, PASSWORD_
DEFAULT);
```

Pour vérifier l'exactitude d'un mot de passe fourni par un utilisateur, vous utiliserez **password_verify()** :

```
if (password_verify($password,$hash)
{
  echo "Mot de passe correct !";
}else{
  echo "Erreur de mot de passe !";
}
```

Cette fonction est simple à utiliser : elle reçoit comme arguments d'abord le mot de passe tel qu'il a été saisi par l'utilisateur, puis le hachage que vous aurez stocké lors de la création du compte ou du dernier changement de mot de passe.

La nouvelle API fournit encore deux autres fonctions :

- **password_get_info** : cette fonction prend en argument un hachage de mot de passe et retourne des informations sur la méthode utilisée pour le produire, à savoir l'algorithme utilisé et les options avec lesquelles il a été appelé,
- **password_needs_rehash** : cette fonction analyse un hachage de mot de passe et vérifie s'il a été effectué avec un algorithme et ses options définis en paramètres ; l'idée est de permettre une migration progressive d'un algorithme à un autre plus performant : lorsque l'utilisateur s'identifie, vous vérifier si l'algorithme de hachage utilisé est à jour, si non, vous utilisez le mot de passe qui vient d'être saisi pour mettre à jour le hachage avec le nouvel algorithme. Bien sûr, tout ceci n'est à effectuer que si le mot de passe correspond.

5 Nouveautés à connaître

Passons maintenant rapidement en revue quelques nouveautés moins essentielles, mais qui peuvent tout de même se révéler utiles.

5.1 Déréférencement des tableaux et des chaînes

Le déréférencement est une notion venue du C et qui n'avait jusqu'alors pas lieu d'être en PHP. En C, le déréférencement désigne la possibilité d'accéder au contenu pointé par un pointeur, à l'aide de l'opérateur « * »... Mais, hors une certaine parenté conceptuelle, le déréférencement en C n'a que peu de choses en commun avec cette nouvelle possibilité offerte par PHP.

Pour le développeur PHP, le déréférencement des tableaux et des chaînes de caractères consiste simplement en une nouvelle syntaxe, plus brève, permettant d'accéder plus facilement à un élément du tableau ou de la chaîne, sans passer par une variable les contenant.

Concrètement, il est maintenant possible d'écrire :

```
echo ["GNU/Linux Magazine","Linux Pratique","Linux Pratique Essentiel"][0]."\n";
echo "GNU/Linux Magazine"[2]."\n";
```

ce qui affichera :

```
GNU/Linux Magazine
U
```

Rien de révolutionnaire là-dedans, j'imagine toutefois que cela peut se révéler pratique pour élaborer des chaînes des caractères destinées à l'instruction honnie **eval()**.

5.2 ::class

Il est maintenant possible d'accéder au nom complet d'une classe, y compris son espace de nom, à travers le la propriété statique réservée **class**. Le résultat sera le même que celui donné par la fonction **get_class**. Ce qui signifie que si vous utilisez **self::class**, vous obtiendrez le nom de la classe dans laquelle la méthode a été définie, et non le nom d'une éventuelle classe qui aurait hérité de cette méthode. Pour obtenir le nom de la classe actuellement utilisée dans une méthode dont elle aurait hérité, il faut utiliser la fonction **get_called_class**. Ainsi le code :

```
<?php
namespace taophp;
class test0 {
  static function test() {
    print get_class()."\n";
    print self::class."\n";
    print get_called_class()."\n";
  }
}
class test1 extends test0 {}
test1::test();
```

affichera :

```
taophp\test0
taophp\test0
taophp\test1
```

5.3 empty()

L'instruction **empty()** n'acceptait jusqu'alors comme argument que des variables. Aujourd'hui, ce n'est plus le cas et il est possible de lui passer une expression arbitraire, comme une fonction :

```
empty(ma_fonction()); // maintenant, cette ligne de code est valide !
```

5.4 Parcours de tableau de tableaux avec foreach

Enfin, PHP 5.5 apporte la possibilité de parcourir les valeurs de tableaux contenus dans un autre tableau directement en permettant l'utilisation de la fonction **list** dans l'initialisation d'un boucle **foreach**. Le plus simple est sans aucun doute d'examiner un exemple :

```
$tableaux = [
  [1,2,3],
  [4,5,6],
];
foreach ($tableaux as list($a,$b)) {
  echo "A : $a; B : $b\n";
}
```

Ce script va produire la sortie suivante :

```
A : 1; B : 2;
A : 4; B : 5;
```

Cette nouvelle possibilité n'est pas révolutionnaire, sans aucun doute, mais peu se révéler bien pratique à l'occasion.

Conclusion

Nous avons consacré cet article aux nouvelles fonctionnalités les plus remarquables pour vous donner un avant-goût de cette version et pour vous motiver à migrer. Toutefois, nous n'avons pas pour autant fait le tour de toute la question, et nous ne pouvons que vous recommander de lire attentivement les notes officielles de migration (http://fr.php.net/migration55) si vous vous décidez à franchir le pas. Bien que la plupart des scripts fonctionnant en PHP 5.4 ne devraient pas rencontrer de difficultés, certaines incompatibilités existent tout de même, et il vaut mieux en prendre connaissance de façon à éviter de perdre du temps en cherchant à comprendre un dysfonctionnement lié. En particulier, on notera la fin du support de Windows XP et 2003 (cela ne nous manquera pas !).

Un dernier point d'actualité avant de se quitter : la série 5.3 de PHP arrive en fin de vie avec la version 5.3.27. Depuis le 11 juillet, il n'y aura plus pour elle que des mises à jour de sécurité durant un an. ■

TU DEVIENDRAS WEB DESIGNER, MON FILS

par Emile iMil Heitor [pour GCU canal historique]

Nous avons récemment [GLMF 166] vu comment créer facilement une application Web avec Flask. S'il s'avère relativement simple, muni de quelques connaissances python, de produire un Web Service fonctionnel, réaliser une application moderne, responsive, mais surtout jolie, peut se transformer en véritable cauchemar pour le quidam dont les connaissances en web design se limitent au strict minimum </Autobiographie>.

Fort heureusement, il existe de nos jours de nombreuses béquilles qui aident les déficients de l'ergonomie que nous sommes et, parmi elles, le module python WTForms et le *framework* Bootstrap qui nous permettrons respectivement de générer des formulaires propres et de produire une interface de belle facture.

1 Au début était le moche

Nous démarrerons notre expérience par la création d'une application Flask basique: Un formulaire de trois champs dont le résultat sera inscrit dans un fichier plat après validation.

Le code Flask est le suivant :

```
$ cat basic.py
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def basicform():
    if request.method == 'POST':
        with open('/tmp/test.txt', 'w') as f:
            for k in request.form:
                f.write('{0}: {1}\n'.format(k, request.form[k]))

    return render_template('basic.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Et le template associé :

```
$ cat templates/basic.html
<!doctype html>
<title>Bienvenue, visiteur du futur !</title>
<div class="content">
    <h2>Identification</h2>
    <hr>
    <form method="POST" action="/">
        <label>Utilisateur</label><br />
        <input type="text" name="user" id="user"><br />
        <label>Mot de passe</label><br />
        <input type="password" name="passwd" id="passwd"><br />
        <select name="active" id="active">
            <option value="no">Non</option>
            <option value="yes">Oui</option>
        </select><br />
        <input type="submit" name="action" value="register">
    </form>
</div>
```

Sans surprise, ce duo produit l'*interface* suivante :

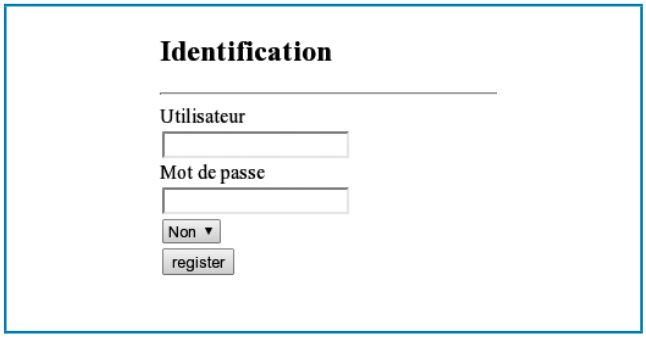


Fig. 1 : Moche

C'est laid, c'est sur, mais surtout, que de caractères tapés pour un résultat aussi sommaire... Alors que nous utilisons un *framework* pour générer notre *GUI*, n'y a-t-il pas de moyen plus efficace pour produire ces sempiternels formulaires ?

2 C'est quoi ce !@#!@#

WTForms est un module *python* permettant de grandement factoriser la création de formulaires. En couplant la puissance du moteur de *template* Jinja2, nativement intégré

à Flask et la souplesse de WTForms, il devient simplissime de générer des formulaires avec un minimum d'efforts.

Il suffit pour cela d'importer le type de champs souhaité, puis de faire hériter son formulaire de l'objet *Form* et d'y lister les différents types de données. Plus fort, WTForms permet sans beaucoup plus de travail d'intégrer des *validateurs*, afin de vérifier que le contenu des champs correspond bien à l'entrée souhaitée; par exemple, en vérifiant le nombre de caractères ou encore la valeur maximale d'une valeur.

Après avoir préalablement installé le module WTForms, par exemple via *pip*, nous créons un fichier **forms.py** contenant les champs du formulaire précédemment créé à la main :

```
from wtforms import Form, TextField, SelectField, SubmitField, PasswordField,
validators

class BasicForm(Form):
    user = TextField(u'Utilisateur', [validators.Length(min=4, max=20)])
    passwd = PasswordField(u'Mot de passe', [validators.Length(min=8, max=32)])
    active = SelectField(u'Actif', choices = [('yes', 'Yes'), ('no', 'No')])
    action = SubmitField(u'Register')
```

On importe cette *classe* dans notre fichier **basic.py** et instancions un objet **form** que l'on passera en paramètre à notre *template*, capable de l'utiliser via Jinja :

```
from flask import Flask, request, render_template, url_for
from forms import BasicForm

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def basicform():
    form = BasicForm(request.form)

    if request.method == 'POST' and form.validate():
        with open('/tmp/test.txt', 'w') as f:
            for k in request.form:
                f.write('{0}: {1}\n'.format(k, request.form[k]))

    return render_template('basic.html', form=form)

if __name__ == '__main__':
    app.run(debug=True)
```

On notera qu'on passe la requête en paramètre à la classe *BasicForm*, ceci afin de contrôler la validité des champs postés via notre formulaire.

Apprécions maintenant la puissance du duo Jinja/WTForms. Voici notre précédent *template*, réduit à un plus simple appareil :

```
$ cat templates/basic.html
<!doctype html>
<title>Bienvenue, visiteur du futur !</title>
<div class="content">
    <h2>Identification</h2>
    <hr>
    <form method="POST" action="/">
    {% for field in form %}
    {{ field.label }}
    {{ field }}<br />
    {% endfor %}
    </form>
</div>
```

Ah oui, ça oui, c'est classe (ah-ah jeu de mots involontaire). Dans ce minuscule template, nous bouclons sur le paramètre **form** que nous avons passé en paramètre et, pour chaque valeur, on affiche la propriété **label**, la chaîne de caractères spécifiée en premier paramètre des champs définis dans le fichier **forms.py**, et le champs lui même. WTForms se charge seul de placer correctement toutes les balises nécessaires et nomme les attributs selon le nom de la variable contenant l'objet à afficher.

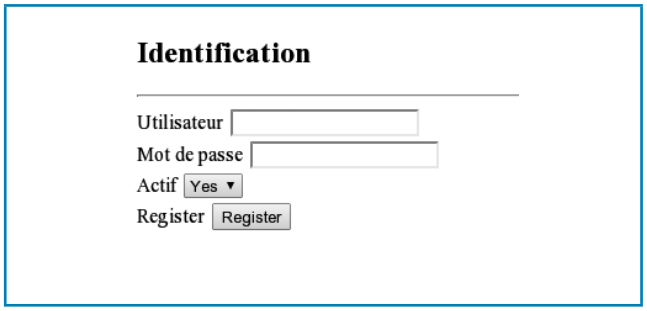


Fig. 2 : Pouah...

3 Et tu nous prends pas pour des jambons des fois ?

Je sais, je sais, c'est toujours moche. Nous y venons.

Bootstrap est un framework libéré en Août 2011 par *Twitter* simplifiant de façon radicale la stylisation d'un site web. Son utilisation est remarquablement aisée et sa documentation limpide et concrète. Le champs d'action de Bootstrap va des formulaires aux barres de navigation, en passant par les labels et autres tableaux. Une particularité notable de Bootstrap, et l'un des aspects qui en font un projet extrêmement populaire, réside dans sa capacité native à gérer le Responsive Design, ou en termes moins *flanbyesques*, la capacité d'un site à s'adapter à tout type de périphériques: stations de travail aux écrans sur-dimentionnés, tablettes, *smartphones*... Cette fonctionnalité repose sur un concept clair et intelligent, Bootstrap découpe la fenêtre du navigateur en une grille de douze colonnes; en ajoutant à vos balises une classe de type **col-<type>-<taille>**, on sait exactement comment se positionnera l'élément dans la page.

Exemple :

```
<div class="col-xs-12 col-md-8">.col-xs-12 col-md-8</div>
```

Ici, pour un périphérique mobile, bénéficiant donc d'une taille réduite (**xs** pour *extra small*), on spécifie que l'élément ** ** occupera 12 colonnes alors que pour un équipement de taille moyenne (**md** pour medium), il n'en occupera que 8.

De façon plus générale, la stylisation de tous les éléments d'un site web à l'aide de ce *framework* s'effectue en ajoutant des classes aux éléments souhaités, on rendra par exemple un

bouton *beau* en lui appliquant les classes **`btn btn-default`** pour un design *standard* ou encore **`btn btn-primary`** pour un bouton sur lequel on veut mettre l'emphase.

Dans notre *template*, on « active » Bootstrap de façon triviale. Après avoir téléchargé l'archive de Bootstrap, on copie les fichiers **`dist/css/bootstrap.min.css`** et **`dist/css/bootstrap-theme.min.css`** dans le répertoire **`static`** de notre projet Flask. Il s'agit ici des versions *minifiées*, privées des retours à la ligne et espacements inutiles. Nous ne nous occupperons pas, dans cet article, des capacités *JavaScript* de Bootstrap, aussi nous ne copierons pas le fichier **`dist/js/bootstrap.min.js`**.

Voici l'allure de notre *template bootstrapé* :

```
<!doctype html>
<html>
  <head>
    <title>Bienvenue, visiteur du futur !</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel=stylesheet type=text/css href="{{ url_for('static',
filename='bootstrap.min.css') }}" media="screen">
    <link rel=stylesheet type=text/css href="{{ url_for('static',
filename='bootstrap-theme.min.css') }}" media="screen">
  </head>
  <body>
    <div class="content" style="padding: 10px; width: 300px;">
      <form method="POST" action="/">
        <fieldset>
          <legend>Identification</legend>
          {% for field in form %}
            <div class="form-group">
            {% if field.type == 'SubmitField' %}
              {{ field(class="btn btn-primary") }}
            {% else %}
              {{ field.label }}
              {{ field(class="form-control") }}
            {% endif %}
            </div>
          {% endfor %}
        </fieldset>
      </form>
    </div>
  </body>
</html>
```

L'en-tête de ce *template* est très largement inspirée de l'exemple proposé sur le site de Bootstrap, que ces derniers recommandent de copier *verbatim*. La seule modification, outre l'absence de l'inclusion de la section *JavaScript*, réside dans l'utilisation de la fonction Flask **`url_for('static')`**, qui renvoie le répertoire contenant les éléments statiques du projet Flask.

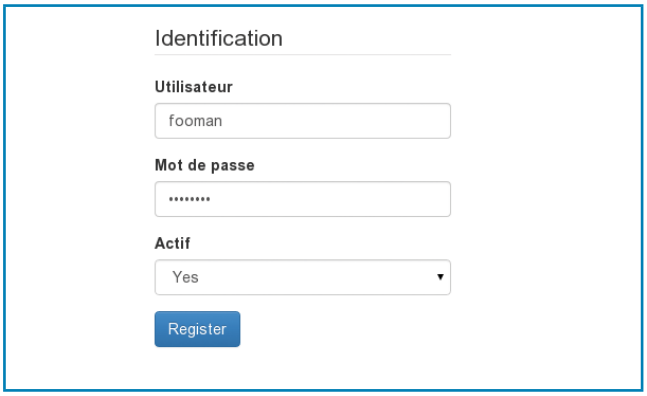


Fig. 3 : oooOOOOOooooh

La structure du formulaire est elle aussi dictée par la documentation de Bootstrap, cette simple organisation et l'ajout de la classe **`form-control`** aux éléments du formulaire a pour immédiate conséquence le rendu suivant : (Fig. 3)

Eh oui, tout ce temps je vous avais caché qu'en fait, je suis un puissant *web designer* r0x0r.

4 Je vois tes yeux briller

Vous imaginez bien que les possibilités du triptique Flask-WTForms-Bootstrap sont très vastes, pensez qu'en écrivant les *macro*s Jinja adéquates, on peut encore factoriser de façon drastique une application web, si complexe soit-elle. Voici deux exemples de *macro*s que j'utilise dans un projet professionnel articulé autour des composants que nous avons expérimenté :

```
{% set bs_col = 'col-sm-8' %}

{% macro input_text(field) -%}
  <div class="form-group {{ bs_col }}">
    {{ field.label }}
    {% if 'dsc' in field.description %}
      <small><em>( {{ field.description['dsc'] }} )</em></small>
    {% endif %}
    {{ field(class="form-control", placeholder=field.description['ph']) }}
  </div>
{%- endmacro %}

{% macro select(field) -%}
  <div class="form-group  {{ bs_col }}">
    {{ field.label }}
    {{ field(class="form-control") }}
  </div>
{%- endmacro %}
```

Il est alors possible d'appeler des macros au sein d'autres templates Jinja grâce à la capacité de ce dernier module à importer des fichiers, tels que son langage originel *python*.

Pour finir, la courbe d'apprentissage des solutions présentées est relativement courte, essentiellement grâce aux excellentes documentations proposées par ces projets (tous Libres, je le rappelle), profitez donc de ces technologies modernes, pourvues d'un rendu exemplaire, qui ne manqueront certainement pas d'épater la galerie. ■

Liens

[1] http://flask.pocoo.org/

[2] http://fr.wikipedia.org/wiki/Responsive_Web_Design

[3] http://wtforms.simplecodes.com

[4] http://getbootstrap.com/

[5] http://jinja.pocoo.org/

SERVEURS DÉDIÉS
PREMIÈRE MONDIALE CHEZ 1&1

TOUT NOUVEAU ET DÉJÀ CHEZ 1&1 :

INTEL® ATOM™

1&1 SERVEUR DÉDIÉ A8i

À partir de

39,99 € HT/mois
47,99 € TTC*

NOUVEAU : 1&1 SERVEUR DÉDIÉ A8i AVEC 30 % DE PERFORMANCE EN PLUS

- Intel® Atom™ C2750
- 8 Cœurs et 8 Go de RAM
- 2 x 1 To SATA HDD
- Parallels® Plesk Panel 11
- Linux, Windows ou Clé-en-main
- Bande passante 100 Mbps
- Architecture 64 bits
- System-on-Chip (SoC) : 30 % de performance supplémentaire

MEMBRE DE united internet

0970 808 911
(appel non surtaxé)

1 MOIS POUR ESSAYER

1 CHOIX VOTRE DURÉE D'ENGAGEMENT

1 APPEL UN EXPERT VOUS RÉPOND

1&1

1and1.fr

* Le serveur dédié A8i est à partir de 39,99 € HT/mois (47,99 € TTC) pour un engagement de 24 mois. Également disponible avec une durée d'engagement de 12 mois ou sans durée minimale d'engagement. Frais de mise en service : 49 € HT (58,80 € TTC). Conditions détaillées sur 1and1.fr. Intel, le logo Intel, Intel Atom et Intel Inside sont des marques commerciales d'Intel Corporation aux États-Unis et/ou dans d'autres pays.

VOS APPLICATIONS WEB TEMPS RÉEL FACILEMENT AVEC METEORJS

par Benoît Benedetti
[Administrateur Système Linux - Université de Nice Sophia Antipolis]

Vous avez depuis des années l'idée d'une application web mais n'avez pas l'envie, le temps de coder la partie dynamique client et d'apprendre le PHP, Python ou Ruby pour développer la logique serveur. Dans cet article, je vais vous présenter MeteorJS, un framework qui va vous permettre de concrétiser vos aspirations de développeur web.

1 Introduction

MeteorJS est un Framework qui permet de créer des applications rapidement. Ses particularités :

- temps réel par défaut, les clients se synchronisant automatiquement par rapport aux données serveur,
- 100 % Javascript, du serveur au client,
- la même API pour développer sur le client, comme le serveur,
- des modifications du code, la page de votre navigateur est automatiquement mise à jour, sans avoir à la recharger,
- réactif : les pages sont également automatiquement mises à jour aux changements des données, de session comme de base de données,
- mise à jour transparente : vous pouvez mettre à jour votre application en production, sans pour autant déconnecter et déranger vos clients,
- une commande qui permet de démarrer rapidement un projet, de manipuler vos données ou de déployer votre application.

Rajoutez à cela un modèle de développement simple, dans lequel il n'y a pas besoin de gérer de l'ajax ou de faire des manipulations complexes du DOM pour ajouter du dynamisme à votre application.

Tout ceci est rendu possible par l'association de plusieurs technologies : NodeJS pour utiliser du Javascript côté serveur. Côté client, le moteur de template HandlebarJS est utilisé pour gérer le rendu html. Pour gérer les données, la base de données orientée documents MongoDB est utilisée. Cela permet de facilement manipuler des données au format JSON pour Meteor.

Meteor implémente le Design pattern Observateur (Observer pattern), suivant lequel les clients se mettent à jour en fonction du serveur. Pour cela, les clients s'abonnent au serveur qui leur envoie des messages au format JSON via le protocole maison DDP **[DDP]**. Pour établir la connexion, les clients utilisent un websocket, si le navigateur est récent et le permet. Si cela n'est pas possible, un client va utiliser la technique AJAX du Long Polling, moins performante, mais rétro-compatible.

2 Pour débuter

2.1 Installation

Meteor est disponible pour Linux, Mac et bientôt Windows. Nous utiliserons une distribution Debian dans cet article, pour travailler avec la version 0.6.6.3 de Meteor. L'ensemble des dépendances de Meteor est facilement installable par un script, que l'on peut exécuter via **curl** :

```
$ sudo aptitude install -y curl
$ curl https://install.meteor.com | /bin/sh
```

Le script exécuté, Meteor et toutes ses dépendances sont installés dans un dossier **.meteor**, pour l'utilisateur courant, à la racine de son répertoire personnel.

2.2 Démarrez un projet

Vous avez désormais à disponibilité l'exécutable **meteor**. Commencez par démarrer un nouveau projet et une nouvelle application avec la commande **create** de **meteor** :

```
$ meteor create glmf-app
```

Cette commande a généré l'application **glmf-app** dans un dossier éponyme :

```
$ cd glmf-app
$ ls
glmf-app.css  glmf-app.html  glmf-app.js
```

Une application basique a été générée, avec trois fichiers : un fichier de style css(vide), un fichier html glmf-**app.html** et le code javascript client et serveur dans un fichier commun **glmf-app.js**. Nous reviendrons par la suite sur le contenu de ces différents fichiers. Pour l'instant, démarrez le serveur pour pouvoir visualiser votre application(tapez cette commande dans un deuxième terminal) :

```
$ meteor
=> Meteor server running on: http://localhost:3000/
```

Comme indiqué dans la sortie de la commande, vous pouvez visualiser cette application de base en ouvrant avec votre navigateur l'url http://ip.de.votre.serveur:3000 (Figure 1). Nous reviendrons plus loin sur cette application. Commençons par découvrir les possibilités de la ligne de commande.

Fig. 1

2.3 Paquets

L'exécutable **meteor** permet de gérer les paquets installés pour votre application. Un paquet Meteor permet d'ajouter facilement une fonctionnalité à votre application : support de Jquery, de l'authentification, etc.. Ces paquets sont développés et maintenus par l'équipe officielle de Meteor et disponibles sur un dépôt officiel. Pour lister les paquets disponibles :

```
$ meteor list
accounts-base          A user account system
...
```

Pour lister les paquets installés d'un projet Meteor, depuis le dossier du projet, utilisez l'option **--using** de la commande **list** :

```
$ meteor list --using
standard-app-packages
autopublish
insecure
preserve-inputs
```

Ces quatre paquets sont installés de base pour tout nouveau projet. Nous reviendrons plus tard sur **autopublish** et **insecure**, que nous supprimerons avec la commande **remove**. Les deux autres paquets sont des paquets de base utiles à tout projet Meteor. La commande **add** permet d'installer facilement un paquet depuis le dépôt officiel mais ne permet pas d'utiliser de dépôt tiers, comme le dépôt Atmosphere **[ATMOSPHERE]**, qui contient énormément de paquets très utiles. C'est pourquoi nous allons installer Meteorite, un utilitaire qui permet de gérer les paquets du dépôt officiel, tout comme ceux d'Atmosphere. Meteorite est un module npm, c'est-à-dire un module NodeJS. Il nous faut donc installer NodeJS :

```
$ echo "deb http://ftp.debian.org/debian wheezy-
backports main contrib non-free" | sudo tee /etc/
apt/sources.list.d/backports.list
$ sudo apt-get update
$ sudo apt-get install nodejs-legacy
```

Puis npm :

```
$ curl https://npmjs.org/install.sh | sudo sh
```

Pour enfin pouvoir installer Meteorite :

```
$ sudo npm install -g meteorite
```

On peut ensuite exécuter l'utilitaire **mrt** de Meteorite pour installer le paquet du framework de style Bootstrap, par exemple :

```
$ mrt add bootstrap
```

Meteor va installer le paquet, puis charger automatiquement les fichiers de Bootstrap dans votre projet. La page web va être automatiquement mise à jour sans avoir à la recharger (ici, Bootstrap va changer la police de caractère du texte affiché).

2.4 Déploiement

La commande **deploy** de meteor permet de déployer et gérer en production votre application facilement et rapidement sur les serveurs du projet Meteor. Cette commande prend en argument le sous domaine du nom DNS meteor.com auquel vous voulez que votre application soit disponible. Ainsi :

```
$ meteor deploy glmf-app
```

Va minifier et empaqueter les fichiers de votre projet, uploader ce paquet sur les serveurs de Meteor puis rendre disponible votre application à l'adresse glmf-app.meteor.com.

Cet environnement de déploiement est encore très limité et ne fournit pas par exemple d'accès ssh pour administrer votre application. Vous pouvez par contre utiliser la commande suivante pour afficher les logs de votre application distante :

```
$ meteor logs glmf-app
```

Ou encore ouvrir un shell dans la base MongoDB de votre application distante (base de données vide par défaut) :

```
$ meteor mongo glmf-app
```

Vous pouvez protéger par un mot de passe votre site lorsque vous utilisez les trois commandes **deploy**, **logs** et **mongo**, pour éviter les actions malveillantes. Pour cela, passez l'option **--password** ou **-P** à **deploy** :

```
$ meteor deploy -P glmf-app
```

Il vous sera demandé d'initialiser un mot de passe, qui sera demandé à l'exécution des différentes commandes.

Si vous désirez arrêter de publier votre application sur meteor.com, passez l'option **--delete** ou **-D** à **deploy** :

```
$ meteor deploy -D glmf-app
```

Vous n'êtes pas limité à héberger votre application sur les serveurs de Meteor. Le framework Meteor est basé sur NodeJS. Meteor fournit une

commande **bundle** pour empaqueter votre application au format NodeJS :

```
$ meteor bundle /tmp/glmf-app.tar.gz
```

Il vous suffit ensuite de dépaqueter cette archive et de démarrer votre application comme une application NodeJS, en suivant les instructions du fichier explicatif **README**, contenu dans l'archive.

Voilà pour un petit tour de l'exécutable **meteor**. Dans la prochaine partie, nous allons voir les concepts simples de Meteor mis en jeu par l'application par défaut, avant de voir des concepts plus avancés dans la partie suivante, via une nouvelle application.

2.5 Application de base

2.5.1 Dossier .meteor

À la racine de votre projet et de tout projet Meteor, un dossier **.meteor** contient des informations sur votre application : liste des paquets installés, version de Meteor utilisée, etc.. Nous verrons plus loin que ce dossier contiendra également les données de la base MongoDB locale. La commande **mongo** de **meteor** vous permettra de travailler avec cette base locale en ligne de commande.

2.5.2 Template

Notre application possède un fichier html **glmf-app.html** :

```
<head>
  <title>glmf-app</title>
</head>

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

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

Le contenu affiché par notre page est défini entre les balises **<body></body>**, via une expression Handlebars. Handlebars est chargé par Meteor de passer à la moulinette tout fichier html : tout ce qui est entre **{{}}** est évalué par Handlebars, le reste est rendu directement en html. Dans notre balise body, **{{> hello}}**, est une expression qui indique d'utiliser le template (en débutant l'expression par un signe **>**) nommé **hello**. Ce template **hello** est par simplicité déclaré tout de suite après dans ce même fichier html. Il affiche le header h1 **Hello World !** et un bouton, visibles sur la page. Comme vous pouvez vous en douter, le texte **Welcome to glmf-app** affiché sur notre page est généré par l'expression Handlebars **{{greeting}}**. Cette expression n'est pas un appel à un template mais un appel à un helper. Un helper se définit dans le code Javascript de votre application Meteor.

2.5.3 Javascript

La « logique » de notre application est contenue dans le fichier **glmf-app.js** :

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

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

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

Tout ce qui est dans un bloc **Meteor.isClient** est exécuté uniquement dans le navigateur des clients. Le bloc **Meteor.isServer** définit du code à exécuter exclusivement sur le serveur. Tout code qui n'est pas dans un de ces blocs est exécuté à la fois côté client et serveur. Notre application est simple, elle ne contient aucun code commun. Côté serveur, seule la fonction **Meteor.startup** est exécutée, vide. Côté client, on définit un helper pour un template en utilisant une fonction **Template.nomDuTemplate.NomDuHelper**. Comme attendu, le helper **Template.hello.greeting** retourne une simple chaîne de caractères, qui remplacera l'appel **{{greeting}}** du fichier html.

Pour ajouter un événement à un template, on crée une fonction **Template.nomDuTemplate.events**, dans laquelle on définit les actions à lier à l'événement. Ainsi, pour notre client, on a défini une fonction associée au clic sur la balise de type **input**(le bouton). Vous voyez que cette fonction affiche un message dans la console de votre navigateur, c'est pourquoi jusqu'à présent un clic sur ce bouton semblait sans effet. Si vous ouvrez la console de votre navigateur (**F12** sous Chrome, que nous utiliserons intensivement dans la suite de l'article) et cliquez sur le bouton, le message de la méthode **console.log** apparaîtra dans la console (Figure 2).

Dans le bloc **Meteor.isServer** on appelle la fonction **Meteor.startup**, dont l'utilité est d'exécuter du code Javascript dès le lancement du serveur, avant tout autre code. En pratique, un bloc **Meteor.startup** côté serveur est souvent utilisé pour peupler une base de données avec un ensemble d'enregistrements initiaux.

Fig. 2

2.6 Arborescence et chargement

L'application exemple générée est minimaliste, à ce titre tout le code Javascript, client comme serveur, est compris dans le même fichier. Idem pour le code Html et template. Mais vous pouvez structurer de manière plus fine votre projet pour qu'une application plus complexe reste maintenable. Avant de voir quelles conventions proposent Meteor, rappelons au passage que Meteor, par défaut, est destiné à créer des Single Page App (SPA), des applications à page unique. De fait, il n'y a pas à créer plusieurs pages html, accessibles à différentes URL.

Meteor concatène et minifie les différents fichiers en un fichier unique pour exécuter votre application : tout les fichiers html, et leur contenu html et templates, sont concaténés en un fichier avant d'être envoyé au client ; idem pour le css, avant envoi au client ; ainsi que pour le code Javascript, côté client comme serveur. Meteor est particulièrement efficace pour parcourir tous les fichiers de tous les dossiers et sous-dossiers de votre projet pour faire ce travail de concaténation avant envoi au serveur et clients. Certaines règles sont à connaître quant à la manière suivie par Meteor pour l'ordre de chargement de vos fichiers :

- les fichiers les plus profonds dans l'arborescence sont chargés en premier,
- tout code Javascript contenu dans un dossier **`lib/`** sera chargé en premier,
- tout code Javascript contenu dans un fichier dont le nom est **`main.js`** sera chargé en dernier,
- pour des fichiers d'une même priorité, l'ordre de chargement s'appuie sur l'ordre alphanumérique.

À ces règles, s'ajoutent :

- tout code JS contenu dans un dossier **`server`** sera uniquement exécuté sur le serveur,
- tout code JS contenu dans un dossier **`client`**, sera uniquement exécuté sur le serveur,
- tout code JS contenu dans un autre dossier que **`server`** ou **`client`** sera disponible pour le client comme le serveur,
- un dossier **`public`** sera utilisable par le client, dans lequel déposer les fichiers images utiles à votre application.

À titre d'exemple pour mettre en évidence cet ordre de chargement, nous allons supprimer l'application de base et générer une arborescence de dossiers :

```
$ rm -f *
$ mkdir shared client server lib
```

Nous allons créer ensuite différents fichiers Javascript dans ces dossiers :

```
$ echo "console.log('shared/main.js')" > shared/main.js
$ echo "console.log('client/a.js')" > client/a.js
$ echo "console.log('client/b.js')" > client/b.js
$ echo "console.log('server/a.js')" > server/a.js
$ echo "console.log('server/b.js')" > server/b.js
$ echo "console.log('lib/a.js')" > lib/a.js
```

Ces différents fichiers se contentent d'afficher leur nom dans la console à leur exécution, ce qui nous permet de voir quel fichier est exécuté et à quel moment. Nous allons également créer un fichier un peu plus complexe, **`shared/shared.js`** :

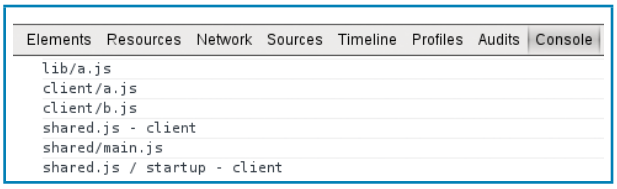


Fig. 3

```
if (Meteor.isClient) {
  console.log('share/shared.js - client');
}

if (Meteor.isServer) {
  console.log('shared/shared.js - server');
}

Meteor.startup(function () {
  if (Meteor.isClient) {
    console.log('shared/shared.js / startup - client');
  }

  if (Meteor.isServer) {
    console.log('shared/shared.js / startup - server');
  }
});
```

Ce fichier n'est pas contenu dans **server** ni **client**, il sera donc exécuté par client et serveur. Nous avons donc utilisé des conditions **Meteor.isServer** et **Meteor.isClient** pour afficher un message caractéristique suivant si le code s'exécute sur le client ou le serveur. Nous avons aussi utilisé un bloc **Meteor.startup** pour mettre en évidence l'effet de cette méthode expliquée dans la section précédente. Redémarrez votre serveur, dans la console l'exécutant vous devriez voir le résultat des affichages générés par l'exécution du code côté serveur :

```
$ meteor
=> Meteor server running on: http://
localhost:3000/
 lib/a.js
 server/a.js
 server/b.js
 shared.js - server
 shared/main.js
 shared.js / startup - server
```

Comme attendu, aucun fichier du dossier **client** n'est exécuté, ni aucune directive **Meteor.isClient**. De plus, les fichiers du dossier **lib** sont exécutés en premier et Meteor a correctement exécuté les différents blocs du même fichier **shared.js**, avant et après **main.js**.

Si vous ouvrez votre application côté client dans votre navigateur, la page est vide car nous n'avons aucun code html. Ce qui nous intéresse de toute façon sont les affichages de la console de votre navigateur, qui devraient être équivalents aux affichages de la console serveur, les fichiers et blocs clients étant exécutés en lieu et place (Figure 3).

Voilà pour le processus de chargement des fichiers. Meteor vous laisse structurer votre application comme bon vous semble et rien ne vous oblige à utiliser un minimum de fichiers possibles ou d'éclater votre code en plusieurs fichiers. Une application Meteor n'aura pas forcément besoin de gérer de manière si fine l'ordre de chargement et nous resterons simples dans l'application exemple que nous allons aborder dans la suite. Mais cette connaissance pourra vous être utile pour votre application future.

3 Notre Application

Dans la suite, nous allons développer (roulements de tambour)... une application de chat ! Et oui, sans réelle surprise, le développement d'un chat étant un peu le Hello World dans le monde des frameworks temps réel. Même si dans notre cas, ce sera plutôt un Hello Micronesia, notre application exemple étant très limitée (Figure 4) : un utilisateur pourra envoyer un message à un autre utilisateur en cliquant sur son nom. Une liste des messages reçus ou envoyés par l'utilisateur est ensuite affichée. Notre application nécessitera aux utilisateurs d'être authentifiés pour pouvoir participer au chat.

Commençons par faire du ménage dans le répertoire **glmf-app** de notre application Meteor :

```
$ rm -rf *
```

Notre application, par souci de concision et pour se focaliser sur Meteor, n'est pas très sexy. Elle ne comportera donc pas de fichiers de style css. Elle ne comportera pas non plus de fichiers images, nous n'aurons donc pas besoin de dossier **public**. Nous aurons donc seulement besoin des dossiers **server**, **client** et d'un dossier **collections**, dans lequel nous définirons plus tard les modèles de données :

```
$ mkdir server client collections
```

Fig. 4

3.2 Mise en page

Nous allons définir la mise en page globale de notre application dans un fichier **client/glmfApp.html** :

```
<head>
 <title>Ma super application Meteor</title>
</head>

<body>
 {{loginButtons}}
 {{#if currentUser}}
   <h3>Utilisateurs</h3>


   <h3>Messages</h3>

 {{/if}}
</body>
```

Plus tard, notre application fera appel à des templates pour afficher la liste des utilisateurs et des messages, sous les balises **h3** correspondantes. En l'état, dans cette mise en page, on affiche ces h3 seulement si l'utilisateur est authentifié. Ce test est fait via **currentUser**, qui est un helper fourni par le paquetage **accounts-password** qui permet de gérer des utilisateurs. Le helper

loginButtons est fourni quant à lui par le paquetage **accounts-ui**, qui offre des éléments graphiques pour afficher des formulaires de connexion, création de comptes utilisateurs,etc..

3.3 Authentification

On commence par installer le paquet **accounts-ui** pour afficher un menu de connexion/création de compte à la place de l'appel à **loginButtons** :

```
$ mrt add accounts-ui
```

Une fois installé, un message d'erreur rouge apparaît sur votre site, vous indiquant qu'aucun service de connexion n'est configuré (*No login services configured*). Via des paquets supplémentaires, Meteor permet d'utiliser plusieurs services d'authentification : github, twitter, etc.. Nous resterons simples (et libres ;)), en utilisant, comme indiqué précédemment, le paquet **accounts-password**, qui gère en interne l'authentification et les mots de passe des utilisateurs de notre application :

```
$ mrt add accounts-password
```

Ce paquet installé, le helper **loginButtons** fonctionnera et notre application affiche un menu **Sign in**, qui permet d'afficher notre formulaire de connexion/création de compte utilisateur (Figure 5). D'ailleurs, par défaut, le paquet **accounts-password** demande un email à la création et la connexion. Nous allons configurer ce paquet pour utiliser seulement un nom d'utilisateur **[ACCOUNTS-UI-CONFIG]**. Cette configuration sera placée dans un sous-dossier **client/lib** :

```
$ mkdir -p client/lib
```

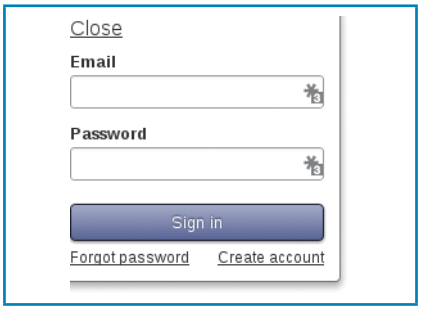


Fig. 5

Le fichier de configuration **client/lib/users.js** contient :

```
Accounts.ui.config({
  passwordSignupFields: 'USERNAME_ONLY'
});
```

En arrière plan, ce deuxième paquet a surtout mis en place une collection Meteor pour gérer nos utilisateurs.

3.4 Collections

3.4.1 Collections Meteor.Users

Une collection est un mécanisme de gestion de données central dans Meteor, qui permet de synchroniser les données contenues dans la base Mongo DB entre les clients/navigateurs et le serveur. Le paquet **accounts-password** a créé pour nous une collection **Users** pour gérer les données de nos utilisateurs enregistrés. Vous pouvez accéder à un shell mongoDB de la base de votre application avec la commande suivante :

```
$ meteor mongo
MongoDB shell version: 2.4.6
connecting to: 127.0.0.1:3002/meteor
>
```

Vous pouvez retourner les utilisateurs en base avec la commande suivante :

```
> db.users.find();
{ "_id" : "BucAfArAMTFjjG58Y", "createdAt" : ISODate("2013-12-06T22:35:48.301Z"), "services" : {
"password" : { "srp" : { "identity" : "9Q2nA6RQ568aDmLFd", "salt" : "R8E9CXxvnEGfmyAEN", "verifier" :
"29f9362989a0c06dbf59e67ac6a10773e45bdee2aef0dea1d9bd43a15f36974742c0ee0655
c05d7afcf0d160741b5f8e6794f49514a7bc1d098eb9e3e8c384ea49cbe6d5ffc84a088fccbdde8cfb1fca79cb820a16b74db22e38ada
2a0c707813bbf65d7817d7c895691aa506fb32141426ebb13fa86e79dbf8dfb2241608 50f" } }, "resume" : { "loginTokens"
: [     {       "token" : "3MoKDuNzhRbjYuyha",    "when" : ISODate("2013-12-06T22:35:48.301Z") },         {
"token" : "Cj9EQGumw24msA8ms",  "when" : ISODate("2013-12-07T11:18:13.790Z") } ] } }, "username" : "bob" }
```

Je ne m'étendrai pas sur la syntaxe pure MongoDB **[MONGODB]**. Ce qui nous intéresse dans cet article est l'API MongoDB de Meteor, équivalente côté client comme serveur. Notre collection **Users** est accessible côté client comme serveur via l'objet **Meteor.Users**. Si vous ouvrez une console dans votre navigateur, vous pouvez d'ailleurs interagir avec la base de votre application depuis le navigateur. Dans la figure 6, à la différence de la syntaxe pure de MongoDB, **find()** retourne un pointeur (curseur) vers la collection. Il faudra boucler sur ce pointeur pour parcourir les différents éléments ou utiliser directement **fetch()**. Les options de la syntaxe MongoDB, options de sélections, de tri, sur les enregistrements sont disponibles, que ce soit dans un **find()** ou dans un **findOne()**(ne retourne qu'un seul enregistrement), comme dans le dernier exemple de la figure 6.

Fig. 6

Vous pouvez remarquer également qu'une requête dans la console MongoDB retourne plus de propriétés pour un objet de type utilisateur, que la collection

disponible pour Meteor. On remarquera que seules les propriétés **_id** et **username** sont disponibles pour Meteor côté client. D'ailleurs nous aurons besoin d'au moins trois comptes créés pour la suite de l'article. Assurez-vous de créer au moins trois comptes utilisateur et de noter leur identifiant respectif.

3.4.2 Créer sa collection Messages

Notre collection **Users** est particulière, elle a été créée pour nous par le paquet **accounts-password**. Pour notre application nous aurons également besoin d'une collection **Messages** pour stocker les messages envoyés par nos utilisateurs. Elle doit être définie pour le serveur et les clients. Ainsi, au lieu de la définir dans un fichier du dossier client ET dans un fichier du dossier serveur, nous la définissons dans le dossier externe **collections** créé précédemment, qui la rendra disponible automatiquement pour le serveur et les clients. Définissez-la dans le fichier **collections/messages.js** suivant :

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

Meteor va automatiquement charger ce nouveau code, créer une base MongoDB **messages** et la mettre à disposition via une collection **Messages** pour notre serveur et ses clients. À la différence de la collection **Users**, spéciale car créée par un paquet, notre collection **Messages** ne nécessite pas d'être préfixée par Meteor pour y faire appel : pour retourner les éléments de la collection, il suffit de faire **Messages.find().fetch()** par exemple, et non **Meteor.Messages.find().fetch()**. Cette commande ne retourne aucun élément pour le moment, car notre base est vide. Nous allons y ajouter des enregistrements. Définissons d'abord ce qui composera un message et notons que MongoDB est flexible dans le sens que nous n'avons pas eu besoin de déclarer une structure de données à la création de notre collection. Nos messages contiendront quatre propriétés : le texte du message (propriété **content**), l'identifiant de l'expéditeur (**senderId**), celui du destinataire (**recepientId**), et la **date** d'envoi. Dans la console de votre navigateur, vous pouvez utiliser la méthode **insert** sur la collection **Messages** pour insérer des messages en base (Figure 7). Comme indiqué dans la figure 7, utilisez des identifiants d'utilisateurs existants et envoyez plusieurs messages en mixant expéditeur et destinataire.

3.5 Templates

Maintenant que nous avons des messages et utilisateurs en base, on va définir les templates pour les afficher. Modifiez le fichier de mise en page **client/glmfApp.html**, pour utiliser le template **messagesList** et afficher la liste des messages :

```
...
   <h3>Messages</h3>
   {{> messagesList}}
...
```

Nous allons définir ce template dans le fichier **client/messagesList.html** :

```
<template name="messagesList">
 {{#each messages}}
   <div>
     <strong>@{{recepient}}</
strong>:{{content}}, {{sender}}
  </div>
 {{/each}}
</template>
```

Notre template va boucler(**#each**) sur **messages**, qui correspond à l'ensemble des messages en base. À chaque tour de boucle, on a accès aux propriétés du message en cours, ce qui nous permet d'afficher son contenu. Pour afficher l'expéditeur et le destinataire, on a recours à des helpers Javascript **recepient** et **sender**, à définir dans un fichier Javascript côté client. Nous les définissons donc dans un fichier **client/messagesList.js** :

```
Template.messagesList.helpers ({
    messages: function() {
      return Messages.find();
    },

    sender: function() {
      return Meteor.users.findOne(this.senderId).
username;
    },

    recepient: function() {
      return Meteor.users.findOne(this.recepientId).
username;
    }
});
```

On les définit dans un bloc **Template.NomDuTemplateEnCours.helpers**. On commence par le helper **messages** qui nous retourne les enregistrements de la collection Messages. Les helpers **sender** et **recepient** renvoient la propriété **username** de l'expéditeur et destinataire du message courant de la boucle, message courant auquel on a accès via l'objet **this**.

Si vous regardez dans votre navigateur, la liste des messages devrait apparaître. Vous pouvez d'ailleurs rajouter un message en utilisant **Messages.insert** dans la console de votre navigateur (ou depuis un autre navigateur avec un autre utilisateur connecté), la liste devrait se mettre à jour en temps réel. Ceci est dû à la nature réactive des données du template : la collection **Messages** est réactive, c'est-à-dire que son contenu

```
> Messages.insert({content: 'salut!', senderId: "Zz9Lskcy2ozYNCPSz", recepientId: "BucAfArAMTFjjG58Y", date: Date.now()});
  "k8vTsNNCwwto29JHw"
> Messages.insert({content: 'salut!', senderId: "hWCg9S4qcbzvRa3RH", recepientId: "BucAfArAMTFjjG58Y", date: Date.now()});
  "ywtvwuHqkWc5nhgSF"
> Messages.insert({content: 'salut!', senderId: "hWCg9S4qcbzvRa3RH", recepientId: "Zz9Lskcy2ozYNCPSz", date: Date.now()});
  "cwrHHauYJbhr3QTmF"
>
```

Fig. 7

est automatiquement mis à jour sur le client d'après les données serveur. Un template qui inclut une collection réactive (comme le fait notre template via le helper **messages**), écoute les changements de la collection et se met automatiquement à jour au besoin.

Nos messages affichés, nous pouvons suivre le même cheminement pour afficher la liste des utilisateurs. On commence par ajouter l'appel à un template **usersList** dans le fichier **client/glmfApp.html** de mise en page :

```
...
   <h3>Utilisateurs</h3>
   {{> usersList}}
...
```

Template que l'on définit dans un fichier **client/usersList.html** :

```
<template name="usersList">
 {{#each users}}
   <div>{{username}}</div>
 {{/each}}
</template>
```

Dans ce template, on boucle simplement sur **users**, **users** étant un helper qui renvoie tous les utilisateurs. À chaque tour de boucle, on affiche la propriété **username** de l'utilisateur en cours de boucle. On définit ce helper **users** dans un fichier **client/usersList.js** :

```
Template.usersList.helpers ({
    users: function() {
      return Meteor.users.find({_id: { $ne: Meteor.userId()}}, {sort:
{username: 1}});
    }
});
```

Notre helper renvoie tous utilisateurs, minus l'utilisateur connecté (**{_id: { $ne: Meteor.userId()}**), en effectuant un tri alphabétique sur leur nom (**{sort: {username: 1}}**).

3.6 Sécurité

Par contre, vous vous rendez compte qu'en tant que telle, notre application n'est pas très sécurisée. Depuis le navigateur, je peux afficher tous les messages, même ceux pour lesquels je ne suis pas expéditeur ou destinataire. Je peux tout aussi bien insérer des messages avec l'expéditeur de mon choix. Ceci est dû au fait que, par défaut, Meteor publie toutes les données du serveur aux clients et qu'il est en mode **insecure** qui permet aux clients de manipuler ces données sans aucune validation préalable.

3.6.1 Publish/Subscribe

Dans notre template **messagesList**, tout comme nous l'avons fait dans notre template **usersList**, nous pourrions définir le helper **messages** pour ne retourner que les messages auxquels le client a accès. Sauf que depuis la console du navigateur, le client a toujours accès à tous les éléments de la collection. Meteor offre un mécanisme beaucoup plus fin pour gérer les données envoyées par le serveur aux clients, le mécanisme de publication/souscription. Par défaut, pour faciliter le développement, Meteor publie toutes les données serveur vers les clients, auxquelles ils souscrivent. Ceci est dû au paquet **autopublish**. Pour sécuriser votre application, supprimez ce paquet... :

```
$ mrt remove autopublish
```

...et la liste des utilisateurs et messages va disparaître. Il faut donc commencer par indiquer au serveur de publier explicitement les collections utilisateurs et messages. On commence par les utilisateurs, dans un fichier **server/users.js** :

```
Meteor.publish("users", function () {
  return Meteor.users.find();
});
```

Pour les messages **server/messages.js** :

```
Meteor.publish("messages", function () {
  return Messages.find({$or: [{senderId: this.userId}, {recepientId: this.userId}]},
{sort: {date: 1}});
});
```

Dans ce fichier, on ne publie au client que les messages dont l'utilisateur connecté (**this.userId**) est soit (**$or**) l'expéditeur, soit le destinataire. On peut passer ensuite côté client, pour souscrire à ces publications. On commence par les utilisateurs, en ajoutant au fichier **client/lib/users.js** :

```
...
Meteor.subscribe('users');
```

On souscrit à la publication **users** définie sur le serveur, qui nous permet de continuer à utiliser la collection **Meteor.Users** côté client et la liste de nos utilisateurs devraient automatiquement réapparaître. Pour les messages, créez le fichier **client/lib/messages.js** suivant :

```
Meteor.subscribe('messages');
```

La liste des messages devraient s'afficher à nouveau, le client n'ayant désormais seulement accès aux messages le concernant.

3.6.2 Insecure

Il nous reste néanmoins à éviter au client de pouvoir créer des messages en utilisant les paramètres de son choix, sans validation serveur. Ceci est dû au paquet **insecure**, installé par défaut, toujours dans un souci de permettre de développer rapidement. Retirez ce paquet :

```
$ mrt remove insecure
```

Si vous essayez d'insérer un nouveau message depuis la console du navigateur, vous devriez avoir un message d'erreur.

Avant de voir comment définir des règles d'autorisation et de validation(et surtout d'ajouter un champ texte pour envoyer un message autrement que par la console du navigateur :-)), voyons en détail comment Meteor assure la synchronisation des données clients/serveur.

Chaque client possède une copie des données serveurs(généralement seulement une partie si l'auto-publication est désactivée). Lorsque par exemple nous envoyons un message, le navigateur va mettre à jour les données affichées sur la page immédiatement, pour que l'utilisateur ait une application la plus réactive et temps réel possible. Ensuite, le client va contacter le serveur pour lui indiquer les données à mettre à jour. Deux cas de figure s'offrent au serveur : si le changement ou l'ajout de données est validé, le client initiateur du changement garde ses modifications locales, tandis que le serveur change les données serveur de référence et propage ces modifications aux autres clients connectés, qui vont mettre à jour leurs données locales et rafraîchir leur page en temps réel ; si le changement n'est pas validé par le serveur, les données serveur restent intactes, les autres clients ne sont pas contactés et le serveur indique au client initiateur d'invalider le changement, d'effacer les données locales et de régénérer la page.

C'est exactement ce deuxième cas de figure qui se produit depuis la suppression du paquet **insecure** : lorsque vous insérez un message par la console du navigateur, la page est rafraîchie avec le nouveau message local, nouveau message qui n'est pas validé et inséré sur le serveur(car nous n'avons pas encore défini les autorisations côté serveur), puis la page est rafraîchie après l'invalidation des changements locaux. Si vous faites attention, ce double rafraîchissement est très rapide, mais perceptible.

Reste à ajouter des autorisations côté serveur pour insérer des messages. En fait nous n'utiliserons qu'une validation : nous n'avons vu que **Messages.insert** pour manipuler les messages, car notre application est simple, mais il existe d'autres méthodes pour manipuler une collection comme la mise à jour (**Messages.update()**) ou la suppression (**Messages.remove()**). Dans le même esprit, nous n'utiliserons donc qu'une validation sur l'insertion d'un nouveau message. Cette validation est à définir côté serveur, à ajouter dans le fichier **server/messages.js** :

```
Messages.allow({
  insert: function(userId, doc) {
    return (userId && doc.senderId === userId
&& doc.recepientId !== userId) ;
  }
});
```

On inclut les validations pour une collection dans un bloc **NomDeLaCollection.allow**. Ici, on ne s'intéresse qu'à la validation de **Messages.insert**, on définit donc une fonction **insert**. Pour une telle fonction, si la valeur de retour est **null** ou **false**, on ne valide pas l'opération. Fonction qui a accès automatiquement à l'utilisateur connecté côté client(via **userId**) et l'enregistrement MongoDB en cours de validation et ses propriétés(via **doc**). Notre méthode **insert** vérifie donc que notre insertion provient bien d'un client avec un utilisateur connecté, que cet utilisateur est bien l'expéditeur du message et qu'il n'est pas le destinataire.

Une fois notre validation créée, vous pouvez à nouveau insérer un message. Par contre vous ne pourrez plus insérer de message ne respectant pas la validation(dont vous n'êtes pas l'expéditeur par exemple), comme c'était le cas précédemment. Nous allons dans la suite modifier l'interface de notre application pour pouvoir envoyer des messages sans passer par la console navigateur.

3.7 Sessions

Pour envoyer un message, nous désirons pouvoir double-cliquer sur le nom d'un utilisateur. Le double-clic va révéler un champ de texte, dans lequel entrer notre message, puis appuyer sur **Entrée** pour l'envoyer. Pour afficher ce champ

Note

La collection Meteor.Users, que les paquets autopublish et insecure soient présents ou non, est déjà sécurisée par défaut Pour cette collection, des validations par défaut évitent qu'un utilisateur puisse modifier les paramètres d'un autre compte. Et une publication par défaut s'occupe de ne publier qu'une partie des informations de l'utilisateur connecté.

de saisie, nous allons utiliser un autre type de données réactives utilisables par Meteor, les sessions. Étant donné que les sessions sont réactives, un template qui manipule une session sera régénéré si la valeur de la session qu'il utilise est modifiée. On commence par modifier le template **client/usersList.html** qui génère la liste utilisateurs :

```
<template name="usersList">
 {{#each users}}
   <div>
     {{#if sending}}
       <input type="text" id="message" />
     {{else}}
       {{username}}
     {{/if}}
  </div>
 {{/each}}
</template>
```

On boucle toujours sur les différents utilisateurs. Dans la boucle, si la condition **sending** est vérifiée, on affiche le champ de saisie du message, sinon on affiche le nom de l'utilisateur courant de la boucle, comme c'était le cas précédemment. La condition fait appel au helper **sending**, que nous devons ajouter au bloc **Template.usersList.helpers** du fichier **client/usersList.js** :

```
Template.usersList.helpers ({
    sending: function() {
      return Session.equals('sending', this._id);
    },

    users: function() {
      return Meteor.users.find({_id: { $ne: Meteor.
userId()}}, {sort: {username: 1}});
    }
});
```

Abonnez-vous !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Consultez l'ensemble de nos offres sur : boutique.ed-diamond.com !

11 Numéros de GNU/Linux Magazine

60€*

au lieu de 86,90 €* en kiosque

Économie 26,90€

Économisez plus de 30%*

* Sur le prix de vente unitaire France Métropolitaine

Nouveauté 2014
TOUS LES HORS-SÉRIES PASSENT EN GUIDE !

Un nouveau format avec une reliure de luxe pour ces Guides de référence de 128 pages à conserver dans votre bibliothèque !

Découvrez tous les Guides déjà parus sur

boutique.ed-diamond.com

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE. Pour les tarifs hors France Métropolitaine, consultez notre site : boutique.ed-diamond.com

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 26,90 €/an ! (soit plus de 3 magazines offerts !)

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur boutique.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

❑ Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.

❑ Je souhaite recevoir les offres promotionnelles de nos partenaires.

GNU LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

Abonnez-vous !

➡ **Tous les abonnements incluant GNU/Linux Magazine :**

* Tarifs France Métro (F) ** Base tarifs kiosque zone France Métro (F)

Consultez l'ensemble de nos offres d'abonnements sur :

boutique.ed-diamond.com

Vous pouvez également commander par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

➡ Nos Tarifs s'entendent TTC et en euros		F	OM1	OM2	E	RM
		France Métro	Outre-Mer		Europe	Reste du Monde
LM	Abonnement GLMF	60 €	75 €	96 €	83 €	90 €
LM+	Abonnement GLMF + GLMF HS (6 Guides)	115 €	147 €	190 €	160 €	173 €
T	Abonnement GLMF + MISC + OS + LP + LE	198 €	253 €	325 €	276 €	300 €
T+	Abonnement GLMF + GLMF HS (6 Guides) + MISC + MISC HS + OS + LP + LP HS (3 Guides) + LE	299 €	382 €	491 €	415 €	448 €

• OM1 : Guadeloupe, Guyane française, Martinique, Réunion, St Pierre et Miquelon, Mayotte

• OM2 : Nouvelle Calédonie, Polynésie française, Wallis et Futuna, Terres Australes et Antarctiques française

MA FORMULE D'ABONNEMENT :

Offre	Zone	Tarif
☐ LM		
☐ LM+		
☐ T		
☐ T+		

J'indique la somme due : (Total) €

Exemple :
Je souhaite m'abonner à l'ensemble des magazines + tous les Hors-séries/Guides et je vis en Belgique. Je coche donc l'offre **T+** (la totale avec tous les Hors-Séries/Guides) puis ma zone (E), le montant sera donc 415 euros.

Je choisis de régler par :

☐ Chèque bancaire ou postal à l'ordre des Éditions Diamond

☐ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

Notre helper **sending** vérifie si une session de nom **sending** existe et si sa valeur est égale à l'identifiant de l'utilisateur courant dans la boucle du **#each**.

On peut tester dans la console si notre helper fonctionne, en utilisant la méthode **Session.set**, pour donner à la variable de session de nom **sending** l'identifiant d'un utilisateur enregistré affiché dans la liste, dont le nom devrait être automatiquement remplacé par un champ de saisie (Figure 8).

Nous allons désormais voir comment utiliser les événements pour manipuler les sessions et afficher le champ de saisie sans passer par la console navigateur pour définir la valeur de la variable de session.

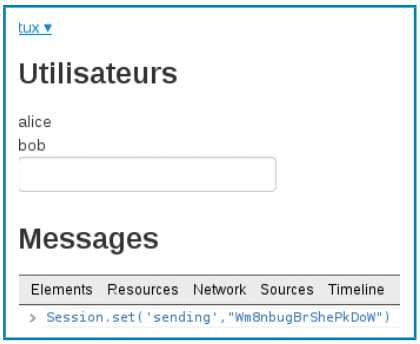


Fig. 8

3.8 Gestion des événements

On désire afficher le champ de saisie via un double-clic sur le nom d'un utilisateur. Pour cela, il faut ajouter un événement, à lier au template dont nous souhaitons modifier le rendu. On ajoute donc le code suivant à notre fichier **client/usersList.js** :

```
....
Template.usersList.events({
  'dblclick': function () {
    Session.set('sending', this._id);
  }
});
```

On lie un événement à un template dans un bloc **Template.NomDuTemplate.events**. On définit ensuite autant d'événements que nécessaire dans ce bloc. Ici, on définit un événement javascript lié au double-clic(**dblclick**) dans une fonction. Cette fonction va appliquer la valeur de l'identifiant de l'utilisateur sur lequel on a cliqué, à la variable de session **sending**.

Une fois ce code ajouté, vous pouvez double-cliquer sur le nom d'un utilisateur pour remplacer son nom par le champ de saisie. Sauf que notre champ ne fait rien pour l'instant, il faut définir un deuxième événement dans ce bloc pour envoyer le message lorsque l'on presse **Entrée** dans le champ de saisie (n'oubliez pas d'ajouter une virgule entre chaque événement défini :

```
Template.usersList.events({
  'dblclick': function () {
    Session.set('sending', this._id);
  },

  'keydown': function (event) {
    if (event.which == 13) {
      var content = document.
getElementById('message').value;

      if (content != '') {
        Messages.insert({
          content: content,
          senderId: Meteor.userId(),
          recepientId: Session.
get('sending'),
          date: Date.now()
        });

        Session.set('sending', null);
      }
    }
  }
});
```

Cet événement est de type **keydown**, il surveille les touches tapées dans le champ de saisie. La fonction liée vérifie si la touche tapée est la touche **Entrée** (code de touche 13). Si c'est le cas, on récupère le contenu du champ de saisie et on insère le message avec ce contenu et les propriétés appropriées. Puis on remet à zéro la variable de session pour effacer le champ de saisie.

Notre application est désormais fonctionnelle. Elle est très loin d'être complète mais nous a permis de voir certains aspects clé du framework Meteor.

Conclusion

Parce-que vous êtes déjà à l'aise en développement web, que vous maîtrisez d'autres framework ou que vous êtes contre le tout-javascript ou encore anti-MongoDB, Meteor n'est sûrement pas fait pour vous. Mais si vous désirez découvrir de nouveaux horizons, développer rapidement un MVP ou que la programmation dynamique n'est pas votre fort, Meteor mérite vraiment votre intérêt.

Il existe d'autres frameworks aux objectifs équivalents, dont DerbyJS. Mais rien ne rivalise actuellement avec l'équipe de MeteorJS (dix ingénieurs à plein temps, sortis du MIT et ayant déjà travaillé sur de gros projets web), une pérennité assurée pendant quelque temps grâce à 11 millions de dollars levés dernièrement pour son développement, mais surtout une communauté déjà importante au vu de la jeunesse du projet, communauté qui ne cesse de croître.

MeteorJS n'est pas encore sortie en version stable mais certaines applications basées sur Meteor sont déjà en production, rien ne vous empêche donc de déjà tester votre idée de prototype avec. La documentation et le blog officiels seront vos premières sources d'informations, les ateliers et présentations tenus mensuellement également **[DEVSHOP]**. Les applications exemples du site officiel **[EXEMPLES]** vous permettront de voir des concepts plus avancés, ainsi que le site **[EVENTMINDED]**, autre source d'inspiration d'intérêt pour développer votre futur projet.. ■

Références

[DDP] https://github.com/meteor/meteor/blob/master/packages/livedata/DDP.md

[ATMOSPHERE] https://atmosphere.meteor.com/

[ACCOUNTS-UI-CONFIG] http://docs.meteor.com/#accounts_ui_config

[MONGODB] http://docs.mongodb.org/manual/crud/

[DEVSHOP] http://www.youtube.com/user/MeteorVideos

[EXEMPLES] http://www.meteor.com/examples

[EVENTMINDED] https://www.eventedmind.com/

COMPILEZ UN NOYAU LINUX

par Thierry GAYET

Le but de cet article sera de comprendre en détail comment générer un noyau GNU/Linux personnalisé. Chaque étape sera reprise et détaillée avec des outils de façon à maîtriser au final l'ensemble.

1 Introduction

La génération d'un noyau GNU/linux suit le même modèle habituellement utilisé dans la communauté open source, à savoir les étapes suivantes :

- récupération des sources,
- configuration des fonctionnalités,
- génération,
- installation,
- lien avec le bootloader.

2 Récupération des sources

2.1 Sources maintenues par chaque distribution

La première solution pour modifier la configuration de son noyau courant est de repartir des sources installées via un package séparé dans le répertoire **/usr/src/**. Chaque version de noyau aura un répertoire préfixé avec **linux-headers** suivi de sa version et de l'extraversion :

```
$ ls -ald /usr/src/linux-headers*
drwxr-xr-x 24 root root 4096 févr. 15  2013 /usr/src/linux-headers-2.6.32-24
drwxr-xr-x 24 root root 4096 févr. 15  2013 /usr/src/linux-headers-2.6.32-45
drwxr-xr-x  7 root root 4096 févr. 15  2013 /usr/src/linux-headers-2.6.32-45-
generic-pae
drwxr-xr-x 24 root root 4096 févr. 15  2013 /usr/src/linux-headers-3.2.0-37
drwxr-xr-x  7 root root 4096 févr. 15  2013 /usr/src/linux-headers-3.2.0-37-
generic-pae
drwxr-xr-x 24 root root 4096 févr. 22  2013 /usr/src/linux-headers-3.2.0-38
drwxr-xr-x  7 root root 4096 févr. 22  2013 /usr/src/linux-headers-3.2.0-38-
generic-pae
drwxr-xr-x 24 root root 4096 mars  19 09:56 /usr/src/linux-headers-3.2.0-39
drwxr-xr-x  7 root root 4096 mars  19 09:56 /usr/src/linux-headers-3.2.0-39-
generic-pae
(...)
```

Pour une version donnée il y a donc bien deux packages : le runtime du noyau et les sources :

```
$ apt-cache search `uname -r`
(...)
linux-headers-3.2.0-52-generic-pae - Linux kernel headers for version
3.2.0 on 32 bit x86 SMP
(...)
linux-image-3.2.0-52-generic-pae - Linux kernel image for version
3.2.0 on 32 bit x86 SMP
```

Le package des sources **linux-headers-3.2.0-52-generic-pae** contient :

```
/usr/src/linux-headers-3.2.0-52-generic-pae
(...)
/usr/src/linux-headers-3.2.0-52-generic-pae/.config
(...)
/usr/src/linux-headers-3.2.0-52-generic-pae/include/media
/usr/src/linux-headers-3.2.0-52-generic-pae/drivers
/firmware
/lib/modules/3.2.0-52-generic-pae/build
```

Alors que le package runtime **linux-image-3.2.0-52-generic-pae** contient :

```
/lib/modules/3.2.0-52-generic-pae
/lib/modules/3.2.0-52-generic-pae/kernel
/lib/modules/3.2.0-52-generic-pae/kernel/crypto
/lib/modules/3.2.0-52-generic-pae/kernel/crypto/algif_skcipher.ko
/lib/modules/3.2.0-52-generic-pae/kernel/crypto/seqiv.ko
(...)
/lib/firmware/3.2.0-52-generic-pae/emi62/bitstream.fw
/lib/firmware/3.2.0-52-generic-pae/emi62/loader.fw
(...)
/boot/System.map-3.2.0-52-generic-pae
/boot/config-3.2.0-52-generic-pae
/boot/abi-3.2.0-52-generic-pae
/boot/vmlinuz-3.2.0-52-generic-pae
```

En même temps que les sources, un fichier de configuration sera fourni :

```
$ ls -al /usr/src/linux-headers-3.2.0-52-generic-pae/.config
-rw-r--r-- 1 root root 147548 juil. 26 19:48 /usr/src/linux-headers-
3.2.0-52-generic-pae/.config
```

Ce fichier est identique à celui fourni pour le noyau couramment chargé :

```
$ ls -al /boot/config-`uname -r`*
-rw-r--r-- 1 root root 147576 juil. 26 19:46 /boot/config-3.2.0-52-generic-pae
```

2.2 Récupération à partir du site web officiel

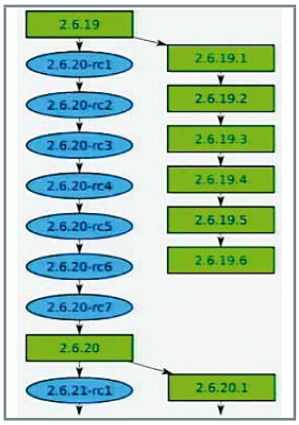
Si on souhaite compiler une version différente du noyau courant, la méthode la plus simple consiste à récupérer les sources d'une version du noyau donné depuis le site web officiel kernel.org.

Ce site ne supporte plus que quatre protocoles contre huit dans le passé : HTTP, FTP et rsync et bien sur GIT.

Comme on peut le voir, le site mentionne plusieurs versions de noyau : stable, EOL ou longterm.

En effet, quand un noyau passe de la branche principale « mainline » à la branche « stable », deux choses peuvent arriver :

- Ils peuvent atteindre une *Fin de vie* après quelques révisions corrections de bugs, ce qui signifie que les mainteneurs du noyau ne publieront plus de corrections de bugs pour cette version de noyau ;
- Ils peuvent être mis en support *à long terme*, ce qui signifie que les responsables continueront à fournir des corrections de bugs pour cette révision du noyau pour une période beaucoup plus longue.



Workflow du développement du noyau GNU/Linux

Si la version du noyau est marquée «EOL», vous devriez envisager de passer à la prochaine version majeure car il n'y aura pas plus de corrections de bugs prévu pour la version du noyau que vous utilisez.

Dans le passé, il était de coutume d'avoir un nombre impair pour représenter les versions de développement (ex : 2.4.17) et paire pour les versions stables (Ex : 2.4.18).

Aujourd'hui, une fois qu'un ensemble de correctifs ont été acceptés, Linux torvald effectuera un freeze de la version ce qui permettra de générer plusieurs Release Candidate (RCXX) qui seront publiées au fur et à mesure sur le site kernel.org. Certains développeurs pourront toujours envoyer leur patchs qui seront intégrés par Linus Torvald pour générer la RC2.

Les prochaines versions RC sont des corrections de bugs et des régressions. Au cours de cette étape, de nombreux développeurs vont tester la nouvelle version et feront rapport de bugs.

Voici un exemple d'évolution de la version 2.6.24 à la version 2.6.25 :

- January 24 : 2.6.24 stable release
- February 10 : 2.6.25-rc1, merge window closes
- February 15 : 2.6.25-rc2
- February 24 : 2.6.25-rc3
- March 4 : 2.6.25-rc4
- March 9 : 2.6.25-rc5
- March 16 : 2.6.25-rc6
- March 25 : 2.6.25-rc7
- April 1 : 2.6.25-rc8
- April 11 : 2.6.25-rc9
- April 16 : 2.6.25 stable release

A noter les fichiers XML et JSON qui sont mis à jour à chaque nouvelle livraison :

- XML : https://www.kernel.org/feeds/kdist.xml
- JSON : https://www.kernel.org/releases.json

Une des méthodes les plus simple est de récupérer l'archive en FTP ou bien en HTTP :

```
$ wget https://www.kernel.org/pub/linux/kernel/v3.x/testing/linux-3.11-rc6.tar.xz
--2013-08-22 16:06:41--  https://www.kernel.org/pub/linux/kernel/v3.x/testing/linux-
3.11-rc6.tar.xz
Résolution de www.kernel.org (www.kernel.org)... 198.145.20.140, 149.20.4.69
Connexion vers www.kernel.org (www.kernel.org)|198.145.20.140|:443... connecté.
requête HTTP transmise, en attente de la réponse... 200 OK
Longueur: 75081388 (72M) [application/x-xz]
Sauvegarde en : "linux-3.11-rc6.tar.xz"
100%[======================>] 75 081 388   535K/s   ds 4m 37s
2013-08-22 16:11:30 (264 KB/s) - "linux-3.11-rc6.tar.xz" sauvegardé
[75081388/75081388]
```

En même temps, il peut être nécessaire de vouloir vérifier que l'intégrité d'une version téléchargée.

Pour ce faire, le site kernel.org publie la clef pgp permettant cette validation :

```
$ wget https://www.kernel.org/pub/linux/kernel/v3.x/testing/linux-3.11-rc6.tar.sign
--2013-08-22 16:07:55--  https://www.kernel.org/pub/linux/kernel/v3.x/testing/linux-
3.11-rc6.tar.sign
Résolution de www.kernel.org (www.kernel.org)... 198.145.20.140, 149.20.4.69
Connexion vers www.kernel.org (www.kernel.org)|198.145.20.140|:443... connecté.
requête HTTP transmise, en attente de la réponse... 200 OK
Longueur: 490 [application/pgp-signature]
Sauvegarde en : "linux-3.11-rc6.tar.sign"
100%[======================>] 490         --.-K/s   ds 0s
2013-08-22 16:08:06 (192 MB/s) - "linux-3.11-rc6.tar.sign" sauvegardé [490/490]
```

Au préalable, il aura fallu télécharger la clef public associée au site kernel.org :

```
$ gpg --keyserver wwwkeys.pgp.net --recv-keys 0x00411886
gpg: demande de la clef 00411886 sur le serveur hkp wwwkeys.pgp.net
gpg: clef 00411886 : clef publique " Linus Torvalds <torvalds@linux-
foundation.org> " importée
gpg: aucune clef de confiance ultime n'a été trouvée
gpg: Quantité totale traitée : 1
gpg:               importées : 1  (RSA: 1)
```

Il est désormais possible de vérifier l'intégrité de l'archive :

```
$ gpg --verify linux-3.11-rc6.tar.sign
gpg: Signature faite le dim. 18 août 2013 23:41:25 CEST avec la clef RSA
d'identifiant 00411886
gpg: Bonne signature de " Linus Torvalds <torvalds@linux-foundation.org> "
gpg: Attention : cette clef n'est pas certifiée avec une signature de confiance.
gpg:          Rien n'indique que la signature appartient à son propriétaire.
Empreinte de clef principale : ABAF 11C6 5A29 70B1 30AB  E3C4 79BE 3E43 0041 1886
```

Si la vérification avait été effectuée avant l'import de la clef publique, nous aurions obtenu le message suivant :

```
$ gpg --verify linux-3.11-rc6.tar.sign
gpg: Signature faite le dim. 18 août 2013 23:41:25 CEST avec la clef
RSA d'identifiant 00411886
gpg: Impossible de vérifier la signature : clef publique introuvable
```

A noter le changement de la clef RSA 0x6092693E par 0x00411886 ce qui a également changeé de référent de « Greg Kroah-Hartman (Linux kernel stable release signing key) <greg@kroah.com> » à « Linus Torvalds <torvalds@linux-foundation.org> ».

Une fois, l'archive vérifiée, nous pouvons procéder à l'extraction :

```
$  tar -Jxvf linux-3.11-rc6.tar.xz && cd inux-3.11-rc6
```

Une autre méthode très convoitée des développeurs est la récupération depuis le repo GIT officiel. Plusieurs repo existent :

- Pour obtenir la dernière release candidate :

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/
linux.git && cd linux
```

- Sinon pour la version stable :

```
$ git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/
linux-stable.git && cd linux-stable
```

Pour connaître la liste des tag disponibles :

```
$ git tag -l
latest
v2.6.11
v2.6.11-tree
v2.6.12
(...)
v3.9.8
v3.9.9
```

Pour passer sur une version spécifique :

```
$ git checkout -b stable v3.9.9
Checking out files: 100% (15510/15510), done.
Switched to a new branch 'stable'
```

2.3 Utilisation de l'outil ketchup

Ketchup est un script écrit en python permettant de passer d'une version de noyau linux à une autre par patch successif. Il permet également d'inclure la vérification des signatures pgp.

Avant de l'utiliser, il faut bien obligatoirement l'installer :

```
sudo apt-get install ketchup python patch bzip2 xz-utils
```

La première étape revient à lister les versions disponibles en téléchargement à l'adresse suivante : http://www.kernel.org/pub/linux/kernel :

```
$ ketchup -l
2.4 (signed)
 2.4 kernel series
2.6 (signed)
 2.6 kernel series
(...)
 Ingo Molnar's realtime-preempt kernel
3 (signed)
 current stable kernel series
3-mm (unsigned)
 -mmotm quilt patchset
3-next (signed)
 3 linux-next tree
3-rc (signed)
 current stable kernel series prereleases
3-rt (unsigned)
 PREEMPT_RT real-time kernel
```

A partir de la version 3.9.9, il est donc possible de basculer à la version 3 de la *release candidate* :

```
$ ketchup -G -r 3-rc
3.9.9 -> 3.11-rc6
Downloading patch-3.9.9.xz
[...]
Applying patch-3.9.9.xz -R
Downloading patch-3.10.xz
[...]
Applying patch-3.10.xz
Downloading patch-3.11-rc6.xz
[...]
Applying patch-3.11-rc6.xz
Current directory renamed to /tmp/linux-3.11-rc6
```

Le paramètre **`--no-gpg`** permet de ne pas effectuer la phase de vérification des signatures. Pour effectué le traitement mais sans application effective, il est possible de passer le paramètre **`--dry-run`** :

```
$ ketchup --no-gpg --dry-run -r 3-rc
3.9.9 -> 3.11-rc6
Downloading patch-3.9.9.xz
Applying patch-3.9.9.xz -R
Downloading patch-3.10.xz
Applying patch-3.10.xz
Downloading patch-3.11-rc6.xz
Applying patch-3.11-rc6.xz
```

3 Définition d'une nouvelle config

Le but de l'étape de configuration est de définir la carte d'identité des fonctionnalités du noyau GNU/linux. Cela activera un certain nombre de fonctionnalités soit en statique dans le noyau monolotique, soit en modules dynamiques.

3.1 Les configurateurs

Pour ce faire, les sources du noyau linux intègre plusieurs type de configurateurs. Certain possédant une interface graphique (ncurses, QT, gtk, ...), il sera nécessaire d'installer plusieurs paquets :

```
$ sudo apt-get install libncurses5-dev qt4-dev-tools  libglade2-dev libglade2-0 libgtk2.0-0 libgtk2.0-dev libglib2.0-0 libglib2.0-dev
```

Le but d'un configurateur est de générer un fichier **.config** qui résumera l'ensemble des fonctionnalités à activer.

L'invocation des configurateurs se fait en passant des noms de targets spécifiques au **Makefile** principale :

- **config** : met à jour la configuration en utilisant une interface en mode texte ;
- **nconfig** : met à jour la configuration en utilisant une interface ncurse (version light en noir et blanc) ;
- **menuconfig** : met à jour la configuration en utilisant une interface ncurse (version couleur) ;
- **xconfig** : met à jour la configuration en utilisant une interface QT4 ;
- **gconfig** : met à jour la configuration en utilisant une interface GTK+2 ;
- **oldconfig** : met à jour la configuration en utilisant le fichier **.config** présent dans les sources du noyau
- **localmodconfig** : met à jour la configuration courante en désactivant les modules dynamiques déchargés ;
- **localyesconfig** : met à jour la configuration en intégrant les modules dynamiques à la partie statique du noyau ;
- **silentoldconfig** : idem que **oldconfig** mais met à jour les dépendances de façon « silentieuse » ;
- **defconfig** : met à jour la configuration en prenant les paramètres par défaut pour une architecture donnée ;
- **savedefconfig** : sauvegarde la configuration courante comme **config**. Minimale par défaut ;
- **allnoconfig** : crée une nouvelle configuration où toutes options sont positionnées à « no » ;
- **allyesconfig** : crée une nouvelle configuration où toutes options sont positionnées à « yes » ;
- **allmodconfig** : crée une nouvelle configuration où toutes options sont activées sous forme de modules ;
- **alldefconfig** : crée une nouvelle configuration où toutes options sont demandées positionnées à leur valeur par défaut ;
- **randconfig** : définit une nouvelle configuration en répondant aléatoirement pour l'activation des options ;
- **listnewconfig** : liste les nouvelles options ;
- **olddefconfig** : idem que **silentoldconfig** mais positionne les nouveaux symboles à leur valeur par défaut ;

Les sources issues d'un tarball issue de kernel.org n'apportent pas de configuration par défaut. Sur une distribution Linux, il est nécessaire de récupérer le fichier de configuration du noyau courant.

Au préalable, il est nécessaire de vérifier la version courante du noyau avec la commande **uname** :

```
$ uname -a
Linux tgayet-desktop 3.2.0-52-generic-pae #78-Ubuntu SMP Fri Jul 26 16:43:19 UTC 2013 i686 i686 i386 GNU/Linux
```

Pour repartir de la configuration du noyau packagé par une distribution donnée, il faut recopier le fichier de configuration **.config** à la racine des sources du noyau linux :

```
$ cp /boot/config-`uname -r`* .config
```

Pour que ce fichier de configuration **.config** soit pris en compte, il faut utiliser la target **oldconfig** ou bien **alldefconfig** :

```
$ make oldconfig
```

A noter que si le noyau courant est compilé avec le pseudo filesystem **/proc/config.gz**, il sera possible de régénérer « à chaud » ce fichier à partir de la config du noyau courante :

```
$ zcat /proc/config.gz > .config
```

Pour ce faire, le noyau doit avoir été compilé avec la configuration suivante :

```
CONFIG_IKCONFIG=y
CONFIG_IKCONFIG_PROC=y
```

Puis activé depuis le menu suivant dans le noyau :

```
General setup --->[*] Kernel .config support[*] Enable access to .config through /proc/config.gz
```

Une fois la phase de configuration terminée, un fichier **.config** doit se trouver à la racine des sources.

Il est à noter que cette fonctionnalité est de moins en mois activée dans les noyau courant.

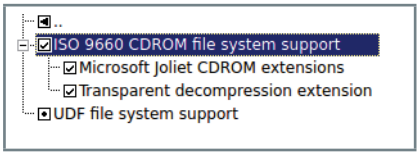
3.2 Détail des menus de configuration

Les menus de la configuration sont organisés en rubriques :

- *Code maturity level options* : Permet de cacher ou de faire apparaître les options qui sont encore en développement et donc considérées comme instables (souvent utile de dire 'oui' ici si l'on veut pouvoir profiter des dernières avancées du noyau) ;
- *General setup* : Ensemble d'options générales sur votre système (sauf si vous voulez compiler pour des architectures très particulières) ;
- *Loadable module support* : Options concernant la gestion des modules (le défaut est presque toujours correct pour une utilisation normale).
- *Block layer* : Les entrées/sorties sur votre carte-mère (peut être supprimé pour un usage embarqué)
- *Processor type and features* : Options relatives au(x) processeur(s): type (x86, Sparc, ...), hyper-thread, dual-core, SMP, etc.
- *Power management options (ACPI, APM)* : Options concernant l'économie d'énergie, la mise en veille et l'ACPI/APM ;
- *Bus options (PCI, PCMCIA, EISA, MCA, ISA)* : Gestion de tous les endroits où vous pourriez enficher des cartes (PCI, PCMCIA, ISA, etc).
- *Executable file formats* : La gestion des fichiers exécutables (Le support ELF doit toujours être à 'Y') ;
- *Networking* : Options concernant les protocoles réseau gérés par votre noyau (le défaut est bien souvent suffisant mais jetez y un coup d'œil à tout hasard).

- *Device Drivers* : Options concernant tous les pilotes matériels (c'est bien souvent ici que l'on passe le plus de temps).
- *File systems* : Options concernant les systèmes de fichiers gérés par votre noyau (vous aurez à y jeter un coup d'oeil).
- *Instrumentation Support* : Option de profilage du noyau (inutile de l'activer).
- *Kernel hacking* ; Options de débogage du noyau (inutile de l'activer sauf si vous avez des envies particulières).
- *Security options* : Options concernant le modèle de sécurité de votre noyau (le défaut est suffisant).
- *Cryptographic options* : Algorithmes cryptographiques pouvant être implantés dans le noyau (le défaut est suffisant).
- *Library routines* : Bibliothèques communes du noyau (le défaut est suffisant).

Chaque menu peut avoir jusqu'à trois états : sélectionné dans la partie statique du noyau (**y**), sélectionné en tant que module dynamique (**m**) et enfin non-sélectionné.



Dans l'exemple du configurateur QT :

- *ISO9660 CDROM file system support* : sera inclus dans un module dynamique **.ko** séparé du noyau ; si cette configuration est sélectionnée, cela activera les deux sous dépendances ;
- *UDF file system support* : sera inclu dans la partie statique du noyau linux

Le fichier de configuration résultant aura au final une ligne par fonctionnalité suivant la règle suivante :

- toute ligne préfixée par un dièze et terminé par « is not set » ne sera pas prise en compte ;
- toute ligne terminée par **=m** générera une module dynamique ;
- toute ligne terminée par **=y** incluera la fonctionnalité dans la partie statique du noyau ;

Exemple de fichier **.config** :

```
(...)
## CD-ROM/DVD Filesystems
#CONFIG_ISO9660_FS=m
CONFIG_JOLIET=y
CONFIG_ZISOFS=y
CONFIG_UDF_FS=y
CONFIG_UDF_NLS=y
## DOS/FAT/NT Filesystems
## CONFIG_MSDOS_FS is not set
# CONFIG_VFAT_FS is not set
CONFIG_NTFS_FS=m
# CONFIG_NTFS_DEBUG is not set
CONFIG_NTFS_RW=y
(...)
```

La configuration du noyau inclus un système de dépendance entre les options. Par exemple, l'activation d'un driver réseau active la pile réseau . Il existe deux types de dépendance :

- Dépendance sur dépendances : une option A dont dépend B reste invisible tant que B n'est pas activé
- Sélection de dépendance : avec une option A dépendant de B, quand A est activé, B est automatiquement activé également. »

make xconfig permet d'afficher toutes les options même celles qui ne sont pas actives. Dans cette version, les options inactives sont juste grisées.

Cette phase de configuration est importante pour choisir les drivers à inclure dans le noyau :

- Il faut éviter de laisser des drivers inutiles car cela augmente la taille du noyau ;
- La part de drivers compilés en statique face aux modules dynamiques doit être prise en compte au sujet du temps de chargement du noyau en mémoire. Il en va aussi de l'initialisation ;
- Trop de drivers compilés en statique inclu une phase de chargement longue sauf si XIP et AXFS sont utilisés (eXecute In Place)

4 Personnalisation de la version du noyau

Avant de lancer la phase de génération, il peut être utile d'identifier le noyau. Un ensemble de variables localisées au début du **makefile** sert à cela :

```
VERSION = 3
PATCHLEVEL = 11
SUBLEVEL = 0
EXTRAVERSION = -rc6
NAME = Linux for Workgroups
```

Cela formera la chaîne suivante : **<VERSION>.<PATCHLEVEL>.<SUBLEVEL> <EXTRAVERSION>**

Exemple obtenu par **uname -r** :

```
$ uname -r
3.2.0-52-generic-pae
```

5 Génération du noyau

5.1 Méthode traditionnelle

Avant de passer à l'étage de génération il faudra vérifier que l'on dispose bien de tous les outils nécessaires :

- GNU C 2.95.3 : **gcc --version**
- GNU make 3.78 : **make --version**
- binutils 2.12 : **ld -v**
- util-linux 2.10o : **fdformat --version**
- module-init-tools 0.9.10 : **depmod -V**
- e2fsprogs 1.29 : **tune2fs**
- jfsutils 1.1.3 : **fsck.jfs -V**
- reiserfsprogs 3.6.3 : **reiserfsck -V 2>&1|grep reiserfsprogs**
- xfsprogs 2.1.0 : **xfs_db -V**
- pcmcia-cs 3.1.21 : **cardmgr -V**
- quota-tools 3.09 : **quota -V**
- PPP 2.4.0 : **pppd --version**
- isdn4k-utils 3.1pre1 : **isdnctrl 2>&1|grep version**
- nfs-utils 1.0.5 : **showmount --version**
- procps 3.1.13 : **ps --version**
- oprofile 0.5.3 : **oprofiled --version**

La génération du noyau GNU/Linux s'effectue en deux phases :

- la partie statique du noyau (**vmlinux**, **bzimage**, ...) ;
- la partie dynamique des drivers (***.ko**)

A noter que la target la plus basique est celle définit par **scripts** :

```
$ sudo make scripts
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf --silentoldconfig Kconfig
  HOSTCC  scripts/genksyms/genksyms.o
  HOSTCC  scripts/genksyms/lex.lex.o
  HOSTCC  scripts/genksyms/parse.tab.o
  HOSTLD  scripts/genksyms/genksyms
  CC      scripts/mod/empty.o
  HOSTCC  scripts/mod/mk_elfconfig
  MKELF   scripts/mod/elfconfig.h
  HOSTCC  scripts/mod/file2alias.o
  HOSTCC  scripts/mod/modpost.o
  HOSTCC  scripts/mod/sumversion.o
  HOSTLD  scripts/mod/modpost
  HOSTCC  scripts/selinux/genheaders/genheaders
compilation terminée.
```

Pour ce faire, un simple **make** permet de lancer la génération en prenant en compte la configuration définit dans le fichier **.config** :

```
$ make
```

Si la machine compilant les sources possède plus d'un processeur ou core, il est possible de paralléliser la génération en utilisant les paramètres **-j** / **--jobs** et **-l** / **--load-average**.

L'option **-j** définit le nombre maximal de tâches en parallèles qui peuvent être exécutées par le **Makefile** et l'option **-l** empêche toute nouvelle tâche parallèle de se lancer à moins que la charge soit inférieure au quota spécifié. La raison pour laquelle le nombre de tâches soit fixé à un plus élevé que le nombre de processeurs est d'aider à assurer la saturation de l'utilisation du processeur.

Pour savoir combien il est possible de paralléliser, un calcul simple est facile à faire :

```
$ export NBCOEUR=$(($(grep -c processor /proc/cpuinfo)+1))
```

ou :

```
$ export NBCOEUR=$(($(cat /proc/cpuinfo|grep processor|wc -l)+1))
```

Pour un INTEL core i7 cela donne 9 et pour un INTEL core i3 cela donne 5.

Pour avoir une génération qui puisse s'adapter de façon optimum à la machine où sont compilés les sources du noyau linux, les commandes suivantes peuvent être utilisées :

```
$ export  NBCOEUR=$(($(cat /proc/cpuinfo | grep processor | wc -l)+1))
$ make -jobs=$((NBCOEUR+1)) --load-average=${NBCOEUR}
```

A noter également que la génération peut être lancée même si l'on ne se trouve pas dans le répertoire des sources. En effet, comme tout **Makefile** GNU, le paramètre « -C » permet de spécifier le path des sources où se trouve le **Makefile** :

```
$ make -j 9 -C=~/linux_src/
```

5.2 Compilation sous debian

Sous ubuntu, le package « kernel package » permet de compiler puis de packager un noyau debian (**.deb**) assez facilement. Cela nécessite une dépendance précise :

```
$ sudo apt-get install kernel-package
```

Puis dans les sources du noyau linux :

```
$ make menuconfig
$ make-kpkg --append-to-version -maversionperso --initrd buildpackage
```

Et après un certain temps dû au build, on peut installer le nouveau noyau à partir du package **linux-image-2.6.38.8-maversionperso_2.6.38.8-maversionperso-10.00.Custom_amd64.deb** :

```
$ ls -al *maversionperso*deb
-rw-r--r-- 1 root root   6294220 nov.  18 20:06 linux-doc-2.6.38.8-
maversionperso_2.6.38.8-maversionperso-10.00.Custom_all.deb
-rw-r--r-- 1 root root   7284416 nov.  18 19:49 linux-headers-2.6.38.8-
maversionperso_2.6.38.8-maversionperso-10.00.Custom_amd64.deb
-rw-r--r-- 1 root root  39522686 nov.  18 19:50 linux-image-2.6.38.8-
maversionperso_2.6.38.8-maversionperso-10.00.Custom_amd64.deb
-rw-r--r-- 1 root root 531472278 nov.  18 20:03 linux-image-2.6.38.8-
maversionperso-dbg_2.6.38.8-maversionperso-10.00.Custom_amd64.deb
-rw-r--r-- 1 root root      2162 nov.  18 20:06 linux-manual-2.6.38.8-
maversionperso_2.6.38.8-maversionperso-10.00.Custom_all.deb
-rw-r--r-- 1 root root  7634010 nov.  18 20:06 linux-source-2.6.38.8-
maversionperso_2.6.38.8-maversionperso-10.00.Custom_all.deb
```

On peut instaler les différents packages :

```
$ sudo dpkg -i *maversionperso*deb
```

On reboot :

```
$ sudo init 6
```

Et dans grub, on choisiera « Previous Linux versions » pour choisir la version avec le suffixe **-maversionperso**.

Une fois logué, on pourra vérifier la version du noyau avec la commande **uname -a** :

```
$ uname -a
Linux PCL131017 2.6.38.8- maversionperso #1 SMP PREEMPT Mon Nov 18
18:15:49 CET 2013 x86_64 x86_64 x86_64 GNU/Linux
```

Et voilà comment, sous Debian/Ubuntu, tester une nouvelle configuration clef en main rapidement.

5.3 Compilation sous Fedora core

La distribution Fedora Core, fournit également une méthode propre et fournissant un RPM au final du même ordre que sous Ununtu.

La procédure reste similaire aux commandes prêt :

```
$ sudo yum install rpmdevtools yum-utils
$ make menuconfig
$ cp .config ~/rpmbuild/SOURCES/config-`uname -m`-generic
$ cd ~/rpmbuild/SPECS
$ vim kernel.spec
```

```
#% define buildid .local

%define buildid .<custom_text>

# cputime accounting is broken, revert to 2.6.22 version
Patch2220: linux-2.6-cputime-fix-accounting.patch
Patch9999: linux-2.6-samfw-test.patch

Again, at the end of the existing patch applications and clearly commented.
ApplyPatch linux-2.6-cputime-fix-accounting.patch

ApplyPatch linux-2.6-samfw-test.patch
```

```
$ rpmbuild -bb --with baseonly --with firmware --without debuginfo  --target=`uname
-m` kernel.spec
$ sudo rpm -ivh $HOME/rpmbuild/RPMS/<arch>/kernel-<version>.<arch>.rpm
```

Pour plus d'information, se référer à la page officielle suivante : ttp://fedoraproject.org/wiki/Building_a_custom_kernel.

6 Modes silencieux ou verbeux

Par défaut la génération des sources est silencieuse, c'est à dire qu'il n'y aura qu'une ligne par action (compilation, édition de lien, ...). Un détail n'étant affiché qu'en cas d'erreur fatale.

Pour passer de ce mode silencieux au mode verbeux, il est possible de changer de mode via le paramètre **V**.

Exemple d'utilisation des deux modes :

```
$ make V=1(pour le mode verbeux)
```

ou :

```
$ make V=0(pour le mode silentieux ; par défaut, donc équivalent à un
simple " make ")
```

A noter aussi l'existence du mode **V=2**. D'autres paramètres comme **C** ou **W** existent.

7 Analyse des fichiers résultant d'un build

Le build permet de générer plusieurs fichiers principaux :

- **./vmlinux** : noyau linux non-compressé utilisé pour le debug
- **./System.map** : table des symboles contenus dans la partie statique
- **./arch/<ARCH>/boot/bzImage** : noyau linux compressé

Le noyau nom compressé est le fichier ELF **vmlinux** qui pourra servir au débug ou bien au profiling. C'est rarement le fichier chargé en ram car on lui préférera une version compressé nommé **bzImage**.

```
$ file vmlinux
vmlinux: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically
linked, not stripped
$ file arch/x86/boot/bzImage
arch/x86/boot/bzImage: Linux kernel x86 boot executable bzImage, version
2.6.35bodino (thierryg@laptopFC, RO-rootFS, swap_dev 0x3, Normal VGA
```

A noter que l'architecture i386 est redirigé vers x86 :

```
arch/i386/boot/bzImage: symbolic link to `../../x86/boot/bzImage'
```

8 Installation

Une fois la phase de génération effectuée, les parties peuvent être installées séparément.

L'installation peut être effectuée en utilisant les paths par défaut ou bien être redéfinis : (voir tableau ci-contre)

L'installation peut être modifiée en précisant un préfixe vers la racine du rootfs. C'est notamment utile pour l'installation du noyau lui même se fait souvent dans le chemin où se trouve le bootloader :

```
$ make INSTALL_PATH=<chemin_noyau> install
```

Pour ce qui est des modules dynamique :

```
$ make INSTALL_MOD_PATH=<chemin_modules_dynamiques> modules_install
```

Pour les firmware :

```
$ make INSTALL_FW_PATH=<chemin_modules_dynamiques> firmware_install
```

Une fois le noyau installé, il faudra demander à **grub** de se mette à jour :

Nom du paramètre	Definition	Chemin par défaut	Target associée
INSTALL_PATH	Spécifie où installer le noyau et le mapping (system map)	/boot	Install
INSTALL_MOD_PATH	Spécifie où installer les modules dynamiques natifs aux sources du noyau	/lib/modules/$(KERNELRELEASE)/kernel	modules_install
INSTALL_MOD_DIR	Spécifie où installer les modules dynamiques extra aux sources	/lib/modules/$(KERNELRELEASE)/extra	modules_install
INSTALL_FW_PATH	Spécifie où installer les firmwares dans l'arborescence des modules dynamiques	$(INSTALL_MOD_PATH)/lib/firmware	firmware_install

```
$ sudo update-grub2
[sudo] password for tgayet:
Generating grub.cfg ...
Found linux image: /boot/vmlinuz-3.5.0-20-generic
Found initrd image: /boot/initrd.img-3.5.0-20-generic
done
```

9 Teste du nouveau noyau

Bon maintenant que le noyau a été généré, il aurait été aussi intéressant de pouvoir le tester dans une machine virtuelle comme **qemu**.

Installons deux packages utiles au préalable :

```
$ sudo apt-get install qemu-system qemu-utils
```

Créons l'image d'un système de fichiers basique sur une image de 800Mo :

```
$ qemu-img create -f raw rootfs.ext2 800M
$ mkfs.ext2 -F rootfs.ext2
$ sudo mkdir -p /mnt/rootfs
$ sudo mount -o loop rootfs.ext2 /mnt/rootfs
$ sudo debootstrap --arch i386 precise /mnt/rootfs http://us.archive.ubuntu.com/ubuntu/
```

Ajoutons les modules dynamiques de notre dernier build :

```
$ sudo make INSTALL_MOD_PATH=/mnt/rootfs/ modules_install
  INSTALL drivers/thermal/x86_pkg_temp_thermal.ko
  INSTALL net/ipv4/netfilter/ipt_MASQUERADE.ko
  INSTALL net/ipv4/netfilter/iptable_nat.ko
  INSTALL net/ipv4/netfilter/nf_nat_ipv4.ko
  INSTALL net/netfilter/nf_nat.ko
  INSTALL net/netfilter/nf_nat_ftp.ko
  INSTALL net/netfilter/nf_nat_irc.ko
  INSTALL net/netfilter/nf_nat_sip.ko
  INSTALL net/netfilter/xt_LOG.ko
  INSTALL net/netfilter/xt_mark.ko
  INSTALL net/netfilter/xt_nat.ko
  DEPMOD  3.12.0-rc2
```

Vérifions la recopie des modules dynamiques :

```
$ tree /mnt/rootfs/lib/modules/
/mnt/rootfs/lib/modules/
├── 3.12.0-rc2
│   └── lib
│       └── modules
│           └── 3.12.0-rc2
│               ├── build -> /home/tgayet/workspace/linux-stable
│               ├── kernel
│               │   ├── drivers
│               │   │   └── thermal
│               │   │       └── x86_pkg_temp_thermal.ko
│               │   └── net
│               │       ├── ipv4
│               │       │   └── netfilter
│               │       │       ├── iptable_nat.ko
│               │       │       ├── ipt_MASQUERADE.ko
│               │       │       └── nf_nat_ipv4.ko
│               │       └── netfilter
│               │           ├── nf_nat_ftp.ko
│               │           ├── nf_nat_irc.ko
│               │           ├── nf_nat.ko
│               │           ├── nf_nat_sip.ko
│               │           ├── xt_LOG.ko
│               │           ├── xt_mark.ko
│               │           └── xt_nat.ko
│               ├── modules.alias
│               ├── modules.alias.bin
│               ├── modules.builtin
│               ├── modules.builtin.bin
│               ├── modules.ccwmap
│               ├── modules.dep
│               ├── modules.dep.bin
│               ├── modules.devname
│               ├── modules.ieee1394map
│               ├── modules.inputmap
│               ├── modules.isapnpmap
│               ├── modules.ofmap
│               ├── modules.order
│               ├── modules.pcimap
│               ├── modules.seriomap
│               ├── modules.softdep
│               ├── modules.symbols
│               ├── modules.symbols.bin
│               ├── modules.usbmap
│               └── source -> /home/tgayet/workspace/linux-stable
```

Avant de se lancer avec qemu, testons le rootfs avec la commande **chroot** :

```
$ sudo chroot /mnt/rootfs /bin/bash
[sudo] password for tgayet:
root@home-gayet:/# ls
bin  boot  dev  etc  home  lib  lost+found
media  mntopt  proc  root  run  sbin
selinux  srv  sys  tmp  usr  var
root@home-gayet:/# exit
exit
```

Vérification l'image crée :

```
$ qemu-img info  rootfs.ext2
image: rootfs.ext2
file format: raw
virtual size: 800M (838860800 bytes)
disk size: 270M
```

Clôturons l'image :

```
$ cd
$ sudo umount /mnt/rootfs
```

Testons enfin le noyau avec le système minimal :

```
$ qemu-system-i386 -kernel workspace/linux-stable/arch/x86/boot/bzImage -s -hda rootfs.ext2 -append root="/dev/sda rw quiet splash --no-log init=/bin/bash"
```

Dans les paramètres pour qemu, on spécifie le noyau, le rootfs ainsi que les paramètres à passer au noyau.

Le noyau se charge en mémoire, s'initialise bien et passe la main au rootfs.

10 Cross-compilation pour l'embarqué

Compiler un noyau linux pour l'embarqué sous-entend générer du code pour une architecture différente de celle de la machine hôte. Pour ce faire, il faut d'une chaîne de cross-compilation, aussi appelée « toolchain » qui a pour rôle de remplir la fonction de ce générateur. Il s'agit en réalité d'un ensemble d'outils comprenant un compilateur, un éditeur de liens, des binutils et tout le nécessaire utile.

Une toolchain est orienté pour une architecture donnée communiqué au **Makefile** via le paramètre **ARCH** (arm, sh4, ...). Il orientera la cross-compilation du noyau en sélectionnant la partie spécifique à chaque architecture, localisée dans le sous-répertoire **arch/** des sources du noyau.

Un listing du répertoire **arch/** permet de se donner une idée du large support du noyau linux :

```
$ ls arch/
alpha  arm    avr32    c6x   frv    hexagon  Kconfig  m68k   microblaze  mn10300
parisc   s390   sh     tile  unicore32  xtensa    arc    arm64  blackfin  cris  h8300
ia64     m32r     metag  mips        openrisc  powerpc  score  sparc  um    x86
```

Le paramètre ARCH pour l'architecture arm sera spécifié de la façon suivante :

```
ARCH =arm
```

Ensuite, les binaires de la chaîne de cross-compilation, possèdent un préfixe permettant tout recouvrement avec deux utilisés pour la machine hôte. Le compilateur de la machine hôte possédera un compilateur **gcc** alors que la toolchain aura **arm-linux-gcc**. Ce préfixe sera définit via le paramètre **CROSS-COMPILE** :

```
CROSS_COMPILE=arm-linux-
```

Un exemple concret serait le suivant :

```
$ make -j5 -C=~/linux_src/ ARCH=sh     CROSS_COMPILE=sh4-linux-
```

ou :

```
$ make -j5 -C=~/linux_src/ ARCH=arm    CROSS_COMPILE=arm-linux-
```

Avant de compiler, la configuration doit se faire également en spécifiant la aussi l'architecture :

```
$ make  ARCH=arm xconfig
```

11 Adaptation du noyau pour le bootloader DAS U-Boot

Uboot nécessite une version adapté du noyau GNU/Linux repackagé spécifiquement pour uBoot. Cette adaptation se fait par la target **uImage** du **Makefile** des sources du noyau Linux via l'utilitaire **mkimage**, non-inclu dans les sources :

Avant de procéder à ce repackaging, il est nécessaire d'installer cet outil :

```
$ sudo apt-get install u-boot-tools uboot-mkimage
```

Ensuite, tout se contrôlera depuis le **Makefile**. Principalement utilisé dans l'embarqué, il faudra aussi spécifé les caractéristiques de la target :

```
$ make ARCH=sh CROSS_COMPILE=sh4-linux-    uImage
```

ou :

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-  uImage
```

12 Autres targets utiles

Le **Makefiles** possède d'autres targets utiles pour nettoyer le projet :

- **clean** : supprime la plupart des fichiers générés mais garde un minimum de support pour générer des modules dynamiques
- **mrproper** : supprime tous les fichiers générés, la configuration et certains fichiers de backup
- **distclean** : effectue ce que réalise **mrproper** et supprime les fichiers de backup (***.bak**, ***.*~**, ...) ainsi que les patchs
- **cleandocs** : supprime tous les fichiers DocBook

13 Aide et documentation

Pour plus d'informations sur les paramètres pouvant être passé au noyau linux, le **Makefile** possède la target **help** qui détaille les paramètres :

```
$ make help
```

Des targets du même **Makefile**, peuvent générer différents types de documentation :

- **htmldocs** : génère les docs au format HTML
- **pdfdocs** : génère les docs au format PDF
- **psdocs** : génère les docs au format Postscript
- **xmldocs** : génère les docs au format XML DocBook
- **mandocs** : génère les man pages noyau
- **installmandocs** : installe les pages de manuel générées par **mandocs**. ■

Liens

- https://www.kernel.org/
- https://git.kernel.org/cgit/
- https://kernel.googlesource.com/
- https://github.com/torvalds/linux
- https://www.kernel.org/signature.html
- http://linux.die.net/man/1/mkimage
- https://www.kernel.org/doc/
- https://www.kernel.org/doc/htmldocs/
- https://www.kernel.org/doc/Documentation/
- https://www.kernel.org/doc/readme/
- http://www.stlinux.com/u-boot/mkimage/kernel-images
- http://www.linuxtopia.org/online_books/linux_kernel/kernel_configuration/
- http://kernelnewbies.org/KernelBuild
- http://www.debian.org/doc/FAQ/ch-kernel.en.html
- http://www.ibm.com/developerworks/library/l-linuxboot/index.html
- http://fedoraproject.org/wiki/Building_a_custom_kernel
- http://kernel-handbook.alioth.debian.org/ch-update-hooks.html#s-kernel-hooks

LANGAGE C

LE GUIDE POUR MIEUX DÉVELOPPER EN C SOUS LINUX !

LES CONSEILS D'EXPERTS POUR MIEUX DÉVELOPPER ET ÉTENDRE VOS CONNAISSANCES DU C

GNU/LINUX MAGAZINE HORS-SÉRIE N°70

DISPONIBLE DÈS LE 03 JANVIER CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR : boutique.ed-diamond.com

A LA DÉCOUVERTE DE DKMS

par Thierry GAYET

Qui n'a jamais eu des problèmes de version de drivers après la mise à jour de son noyau GNU/Linux ? Ce problème est lié principalement à la désynchronisation entre les sources du noyau et celles utilisées pour générer le driver. Nous allons voir comment dkms peu nous aider à régler cela.

1 Introduction

C'est une tendance qui tend à se confirmer, de plus en plus d'industriels fournissent un support dkms à leurs drivers (**virtualbox**, **ndiswrapper**, **lttng**, **fglrx**, ...) de façon à garantir la continuité de service au fil de l'évolution des versions du noyau GNU/Linux mais aussi des drivers eux même.

A chaque changement de version d'un noyau, effectué par le gestionnaire de paquets, un trigger est lancé de façon automatique dans les actions de post-installation des package RPM, DEB, ... La même action est effectuée lorsqu'un drivers est mis à jour.

Il faut donc qu'il soit re-compiler à partir des sources du drivers tout en utilisant l'API du noyau courant garantissant une certaine cohérence.

2 Architecture

DKMS aka Dynamic Kernel Module Support a été développé par la société DELL principalement en langage bash et perl de façon à répondre à cette problématique. Il est distribué librement en open-source sous licence GPLv2.

L'architecture de DKMS est constitué de différentes parties :

1. Les sources du modules : **/usr/src/<drv_name>-<version>/** ;
2. L'arborescence des modules du noyau : **/lib/modules/`uname -r`/** : DKMS y remplacera ou ajoutera les modules que l'on va lui faire gérer ;

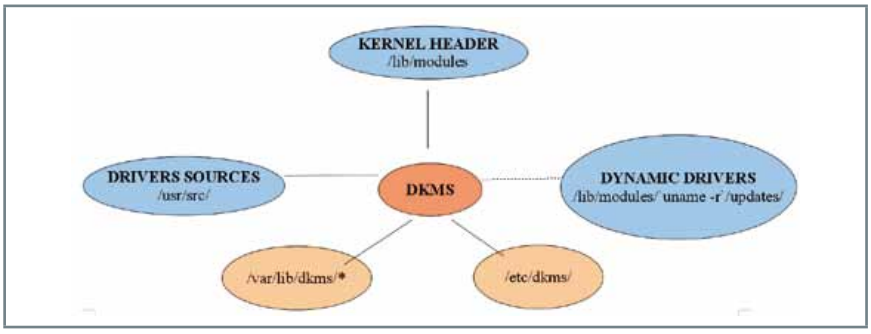


Fig. 1

3. L'arborescence de DKMS **/var/lib/dkms**, contenant :
   - Les répertoires où seront construits les modules (**/var/lib/dkms/<drv_name>/<version>/build/**)
   - Les modules originaux sauvegardés ;
   - Les modules générés par DKMS ;

Fichiers utilisés par DKMS : (Fig. 1)

Concernant DKMS lui même, on y distingue une première partie liée à sa configuration (**/etc/dkms**) et sa partie de travail interne (**/var/lib/dkms/***).

La seconde partie comprend les sources des drivers, les headers par version de noyau GNU/Linux et enfin les modules dynamiques buildés.

3 Installation du cœur de DKMS

Dkms étant devenu une brique incontournable, il est packagé officiellement pour un certain nombre de distributions GNU/Linux :

Debian/ubuntu :

```
$ sudo  apt-get install dkms dh-modaliases
```

OpenSuse :

```
$ sudo zypper install dkms
```

Fedora core :

```
$ sudo yum install dkms
```

Pour les packages les dépendances seront **make**, **linux-headers-generic** et **linux-headers**.

Depuis les sources du repo GIT offciel :

```
$ git clone git://linux.dell.com/dkms.git
```

Par contre, les sources de dkms ne propose pas d'installeur. Les fichiers étant un peu éparpillés un peu partout dans le système de fichiers. Il faut donc recopier les fichiers à la main ; cela reste donc la méthode la moins pratique auquel il faudra lui préférer l'installation par votre système de paquets favoris.

Si on fait une recherche dans les repos, on voit la liste des drivers externes au noyau arriver avec un support dkms augmenter petit à petit au fil des années.

Exemple sous Ubuntu 12.04.3 LTS :

```
# sudo apt-cache search dkms
dh-modaliases - debhelper extension for scanning kernel module aliases
dkms - environnement de gestion dynamique des modules noyau
pvcam-dkms - kernel module for Coolsnap pci card
open-vm-dkms - Source for VMware guest systems driver (DKMS)
sl-modem-source - Pilote des modems logiciels SmartLink - source pour la construction
du module
backfire-dkms - kernel module for signal benchmarking (DKMS)
[...]
virtualbox-ose-guest-dkms - transitional package for virtualbox-guest-dkms
west-chamber-dkms - iptable extension for bypassing content filtering firewall (dkms)
virtualbox - x86 virtualization solution - base binaries
virtualbox-dkms - x86 virtualization solution - kernel module sources for dkms
virtualbox-guest-dkms - x86 virtualization solution - guest addition module source
for dkms
openvswitch-datapath-lts-raring-dkms - Open vSwitch datapath module source - DKMS
version
cedarview-drm - Cedar Trail drm driver in DKMS format.
xtables-addons-common - Extensions targets and matches for iptables [tools, libs]
xtables-addons-dkms - Extensions targets and matches for iptables
```

Une fois installée, la complétion de la commande dkms offres son panel d'options possible :

```
$ dkms
add    build    ldtarball    mkdeb  mkdsc
mkrpm  remove uninstall    autoinstall
install  match    mkdriverdisk  mkkmp
mktarball      status
```

Nous allons voir une partie de ces options dans l'ordre de leur usage.

4 Configuration

Le cœur de DKMS est configurable assez simplement via un fichier de configuration localisé dans **/etc/dkms/framework.conf** dans lequel sont définis tous les path en vigueur dans dkms.

```
## This configuration file modifies the behavior of
## DKMS (Dynamic Kernel Module Support) and is sourced
## in by DKMS every time it is run.
## Source Tree Location (default: /usr/src)
source_tree="/usr/src"
## DKMS Tree Location (default: /var/lib/dkms)
dkms_tree="/var/lib/dkms"
## Install Tree Location (default: /lib/modules)
install_tree="/lib/modules"
## tmp Location (default: /tmp)
tmp_location="/tmp"
## verbosity setting (verbose will be active if you set it to a non-
null value)
# verbose="1"
```

5 Fonctionnement

DKMS fonctionne comme un automate à états. La première étape revient à ajouter le drivers via un fichier de configuration. Ensuite, une fois connu dans la base de données de dkms, il est possible de lui demander de compiler le module sur une ou plusieurs version de noyau GNU/Linux, puis de l'installer dans **/lib/module/<kernel-version>/**...

Chaque étape étant séparée, c'est à dire qu'il est possible d'ajouter un drivers sans qu'il ne soit buildé ou installé.

L'installation pouvant être automatique selon le paramétrage.

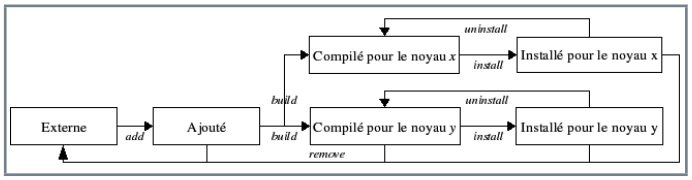


Fig. 2 : Résumé des étapes internes à DKMS

Chaque fichier de configuration s'appuie sur un ensemble de variables propagées par dkms :

- **kernelver** : cette variable contient la version du noyau pour laquelle le module est en train d'être construit. C'est particulièrement utile dans la définition de la commande **MAKE**, par exemple, **MAKE[0]="make INCLUDEDIR=/lib/modules/${kernelver}/build/include"**.
- **dkms_tree** : cette variable indique où se trouve l'arbre DKMS sur le système local. Par défaut il s'agit de **/var/lib/dkms**, mais cette valeur ne doit pas être mise en dur dans les **dkms.conf** au cas ou l'utilisateur aurait changé cela sur son système (Un tel réglage se fait à l'aide de **/etc/dkms/framework.conf** ou en ajoutant l'option **--dkmstree** lors de l'appel).
- **source_tree** : cette variable indique où DKMS stocke les sources des modules sur le système. Par défaut il s'agit de **/usr/src**, mais cette valeur ne doit pas être mise en dur dans les **dkms.conf** au cas ou l'utilisateur aurait changé cela sur son système (à l'aide de **/etc/dkms/framework.conf** ou en ajoutant l'option **--sourcetree** lors de l'appel).
- **kernel_source_dir** : cette variable contient le chemin vers les sources du kernel pour lequel on construit le module. Il s'agit en général de **/lib/modules/$kernelver/build**, à moins qu'un autre chemin n'ait été spécifié avec l'option **--kernel-sourcedir**.

6 Ajout d'un support DKMS pour un driver

Pour montrer comment se fait l'intégration d'un driver externe aux sources du noyau à DKMS, nous allons partir des sources d'un petit driver de caractères simple.

Il sera composé des fichiers suivants dans **/usr/src/monmodule-0.1** :

```
/usr/src/monmodule-0.1
├── dkms.conf
├── monmodule.c
└── Makefile
```

Créons préalablement un répertoire dans **/usr/src/** comprenant le nom du package ainsi que sa version, séparé par un tiret :

```
$ sudo mkdir -p /usr/src/monmodule-0.1
```

Ensuite, ajoutons à ce répertoire, un fichier **dkms.conf** servant à DKMS pour préciser les chemins et nommages des différents éléments associés au driver :

```
$ sudo vim /usr/src/monmodule-0.1/dkms.conf
```

Avec le contenu suivant :

```
PACKAGE_NAME="monmodule"
PACKAGE_VERSION="0.1"
MAKE[0]="make -C ${kernel_source_dir} SUBDIRS=${dkms_tree}/${PACKAGE_
NAME}/${PACKAGE_VERSION}/build modules"
CLEAN=" make -C ${kernel_source_dir} SUBDIRS=${dkms_tree}/${PACKAGE_
NAME}/${PACKAGE_VERSION}/build clean"
BUILT_MODULE_NAME[0]=monmodule
BUILT_MODULE_LOCATION[0]=.
DEST_MODULE_LOCATION[0]=/updates
REMAKE_INITRD="no"
AUTOINSTALL="yes"
```

Pour la target CLEAN, il est possible de faire plus simple :

```
CLEAN="rm -f *.*o"
```

Les sources du module **monmodule.c** étant spécifiée en local (**/usr/src/monmodule-0.1/**) :

```
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <asm/uaccess.h>
MODULE_DESCRIPTION("ex-drv-char-dkms");
MODULE_AUTHOR("tgayet");
MODULE_LICENSE("GPL");
static int major = 0;
module_param(major, int, 0660);
MODULE_PARM_DESC(major, "Static major number (none = dynamic)");
static size_t buf_size = 64;
module_param(buf_size, int, 0660);
MODULE_PARM_DESC(buf_size, "Buffer size");
static char* buffer = NULL;
static size_t num  = 0;
static ssize_t k_read(struct file *file, char *buf, size_t count, loff_t *ppos)
{
  size_t real;
  real = min(num, count);
  if (real)
    if (copy_to_user(buf, buffer, real))
      return -EFAULT;
  num = 0; /* Destructive read (no more data after a read) */
  printk(KERN_DEBUG "char: read %d/%d chars\n", real, count);
  return real;
}
static ssize_t k_write(struct file *file, const char *buf, size_t count, loff_t
*ppos)
{
  size_t real;
  real = min(buf_size, count);
  if (real)
    if (copy_from_user(buffer, buf, real))
      return -EFAULT;
  num = real;
  printk(KERN_DEBUG "char: wrote %d/%d chars\n", real, count);
  return real;
}
```

```
static int k_open(struct inode *inode, struct file *file)
{
  printk(KERN_DEBUG "char: open()\n");
  return 0;
}
static int k_release(struct inode *inode, struct file *file)
{
  printk(KERN_DEBUG "char: release()\n");
  return 0;
}
static struct file_operations k_fops =
{
  .owner =THIS_MODULE,
  .read = k_read,
  .write =k_write,
  .open = k_open,
  .release =k_release,
};
static int __init k_init(void)
{
  int ret;
  ret = register_chrdev(major, "char", &k_fops);
  if (ret < 0) {
    printk(KERN_WARNING "char: unable to get a major\n");
    return ret;
  }
  if (major == 0)
    major = ret;
  buffer = (char *)kmalloc(buf_size, GFP_KERNEL);
  if (buffer != NULL) {
    printk(KERN_DEBUG "char: allocated a %d bytes buffer\n", buf_size);
  } else {
    printk(KERN_WARNING "char: unable to allocate a %d bytes buffer\n", buf_size);
    unregister_chrdev(major, "char");
    return -ENOMEM;
  }
  printk(KERN_INFO "char: successfully loaded with major %d\n", major);
  return 0;
}
static void __exit k_exit(void)
{
  kfree(buffer);
  unregister_chrdev(major, "char");

  printk(KERN_INFO "char: successfully unloaded\n");
}
module_init(k_init);
module_exit(k_exit);
```

Il est aussi nécessaire de joindre un **Makefile** dans le chemin suivant **/usr/src/monmodule-0.1/** :

```
obj-m:= monmodule.o
KDIR := ${kernel_source_dir}
PWD := $(shell pwd)
all:
  make -C $(KDIR) M=$(PWD) modules
install:
  make -C $(KDIR) M=$(PWD) modules_install
clean:
  make -C $(KDIR) M=$(PWD) clean
```

Toujours dans le chemin **/usr/src/mon-module-0.1**, l'ajout du support dkms pour un module se résume à la commande suivante :

```
$ sudo dkms add -m monmodule -v 0.1
Creating symlink /var/lib/dkms/monmodule/0.1/source ->
                 /usr/src/monmodule-0.1
DKMS: add completed.
```

En dehors de ce chemin, il sera nécessaire de spécifier le chemin contenant le fichier de configuration **dkms.conf** via le paramètre **-c** :

```
$ sudo dkms add -m monmodule -v 0.1 -c /usr/src/monmodule-0.1/dkms.conf
Creating symlink /var/lib/dkms/monmodule/0.1/source ->
                 /usr/src/monmodule-0.1
DKMS: add completed.
```

Par défaut, dkms optera un support pour la version du noyau courant, mais pour lui préciser une autre version, le paramètre **-k** peut être utilisé :

```
$ uname -r
3.8.0-32-generic
$ sudo dkms add -m monmodule -v 0.1 -k `uname -r`
Creating symlink /var/lib/dkms/monmodule/0.1/source ->
                 /usr/src/monmodule-0.1
DKMS: add completed.
```

Une fois ajouté, une arborescence est crée dans **/var/lib/dkms/monmodule/0.1/** :

```
$ tree /var/lib/dkms/monmodule/0.1/
/var/lib/dkms/monmodule/0.1/
├── build
└── source -> /usr/src/monmodule-0.1
```

Vérifions aussi le status du module dans dkms concernant ce module :

```
$ dkms status | grep monmodule
monmodule, 0.1: added
```

Pour savoir, d'un point de vue général, les modules qui sont compatibles avec une version spécifique de noyau, la commande **dkms status** supporte le paramètre **-k** auquel il est possible de lui préciser une version de noyau GNU/Linux :

```
$ dkms status -k `uname -r`
backfire, 0.73-1: added
monmodule, 0.1, 3.8.0-32-generic, x86_64: installed
vboxhost, 4.3.0, 3.8.0-32-generic, x86_64: installed
```

Lors d'un ajout, si le module est déjà connu par dkms il indiquera le message d'erreur suivant :

```
$ sudo dkms add -m monmodule -v 0.1
Error! DKMS tree already contains: monmodule-0.1
You cannot add the same module/version combo more than once.
```

7 Génération pour le noyau courant

Vérifions la version de mon noyau GNU/Linux actuellement chargé :

```
$ uname -a
Linux PCL131017 3.8.0-32-generic #47~precise1-Ubuntu SMP Wed Oct 2
16:19:35 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux
$ uname -r
3.8.0-32-generic
```

On peut lister les noyaux actuellement installés :

```
$ dpkg --list | grep linux-image
ii  linux-image-3.8.0-29-generic   3.8.0-29.42~precise1   Linux kernel image for
version 3.8.0 on 64 bit x86 SMP
ii  linux-image-3.8.0-31-generic   3.8.0-31.46~precise1   Linux kernel image for
version 3.8.0 on 64 bit x86 SMP
ii  linux-image-3.8.0-32-generic   3.8.0-32.47~precise1   Linux kernel image for
version 3.8.0 on 64 bit x86 SMP
ii  linux-image-3.8.0-33-generic   3.8.0-33.48~precise1   Linux kernel image for
version 3.8.0 on 64 bit x86 SMP
ii  linux-image-generic-lts-raring 3.8.0.33.33           Generic Linux kernel image
```

Les headers de mon noyau sont localisés dans un répertoire standard :

```
$ ls -al /lib/modules/`uname -r`/
total 4000
drwxr-xr-x  5 root root   4096 nov.  18 09:48 .
drwxr-xr-x  7 root root   4096 nov.  14 12:15 ..
lrwxrwxrwx  1 root root     39 oct.   2 18:50 build -> /usr/src/linux-headers-
3.8.0-32-generic
drwxr-xr-x  2 root root   4096 oct.  22 09:17 initrd
drwxr-xr-x 11 root root   4096 oct.  22 09:17 kernel
-rw-r--r--  1 root root 823213 nov.  18 09:48 modules.alias
-rw-r--r--  1 root root 812020 nov.  18 09:48 modules.alias.bin
-rw-r--r--  1 root root   6404 oct.   2 18:43 modules.builtin
-rw-r--r--  1 root root   8073 nov.  18 09:48 modules.builtin.bin
-rw-r--r--  1 root root     69 nov.  1dkms add -m monmodule -v 0.1 -k `uname -r`
Creating symlink /var/lib/dkms/monmodule/0.1/source ->
                 /usr/src/monmodule-0.1
DKMS: add completed. 8 09:48 modules.ccwmap
-rw-r--r--  1 root root 350671 nov.  18 09:48 modules.dep
-rw-r--r--  1 root root 517082 nov.  18 09:48 modules.dep.bin
-rw-r--r--  1 root root    214 nov.  18 09:48 modules.devname
-rw-r--r--  1 root root   1111 nov.  18 09:48 modules.ieee1394map
-rw-r--r--  1 root root    295 nov.  18 09:48 modules.inputmap
-rw-r--r--  1 root root   4624 nov.  18 09:48 modules.isapnpmap
-rw-r--r--  1 root root   5620 nov.  18 09:48 modules.ofmap
-rw-r--r--  1 root root 144201 oct.   2 18:43 modules.order
-rw-r--r--  1 root root 506787 nov.  18 09:48 modules.pcimap
-rw-r--r--  1 root root   2815 nov.  18 09:48 modules.seriomap
-rw-r--r--  1 root root    131 nov.  18 09:48 modules.softdep
-rw-r--r--  1 root root 328919 nov.  18 09:48 modules.symbols
-rw-r--r--  1 root root 410854 nov.  18 09:48 modules.symbols.bin
-rw-r--r--  1 root root 105714 nov.  18 09:48 modules.usbmap
drwxr-xr-x  3 root root   4096 oct.  22 12:00 updates
```

L'étape de génération du module dynamique pour la version de mon noyau se fait par le paramètre **build** de la commande dkms :

```
$ sudo dkms build -m monmodule/0.1
Kernel preparation unnecessary for this kernel.  Skipping...
Building module:
cleaning build area....
make KERNELRELEASE=3.8.0-32-generic -C /lib/modules/3.8.0-32-generic/
build SUBDIRS=/var/lib/dkms/monmodule/0.1/build modules....
cleaning build area....
DKMS: build completed.
```

La génération peut être fait soit avec le noyau courant (par défaut), soit avec une version spécifique :

```
$ sudo dkms build -m monmodule/0.1 -k `uname -r`
Kernel preparation unnecessary for this kernel.  Skipping...
Building module:
cleaning build area....
make KERNELRELEASE=3.8.0-32-generic -C /lib/modules/3.8.0-32-generic/
build SUBDIRS=/var/lib/dkms/monmodule/0.1/build modules....
cleaning build area....
DKMS: build completed.
```

Le résultat du build aura généré l'arborescence suivante :

```
$ tree /var/lib/dkms/monmodule/0.1/
/var/lib/dkms/monmodule/0.1/
├── 3.8.0-32-generic
│   └── x86_64
```

```
│       ├── log
│       │   └── make.log
│       └── module Arial
│           └── monmodule.ko
├── build
│   ├── dkms.conf
│   ├── Makefile
│   └── monmodule.c
└── source -> /usr/src/monmodule-0.1
```

Pendant cette étape, un fichier de log est crée, nommé **make.log** :

```
$ cat /var/lib/dkms/monmodule/0.1/`uname -r`/x86_64/log/make.log
DKMS make.log for monmodule-0.1 for kernel 3.8.0-32-generic (x86_64)
lundi 18 novembre 2013, 10:01:14 (UTC+0100)
make: entrant dans le répertoire " /usr/src/linux-headers-3.8.0-32-generic "
  CC [M]  /var/lib/dkms/monmodule/0.1/build/monmodule.o
/var/lib/dkms/monmodule/0.1/build/monmodule.c: In function '__check_buf_size':
/var/lib/dkms/monmodule/0.1/build/monmodule.c:16:1: attention : return from
incompatible pointer type [enabled by default]
/var/lib/dkms/monmodule/0.1/build/monmodule.c: In function 'k_read':
/var/lib/dkms/monmodule/0.1/build/monmodule.c:34:2: attention : format '%d'
expects argument of type 'int', but argument 2 has type 'size_t' [-Wformat]
/var/lib/dkms/monmodule/0.1/build/monmodule.c:34:2: attention : format '%d'
expects argument of type 'int', but argument 3 has type 'size_t' [-Wformat]
/var/lib/dkms/monmodule/0.1/build/monmodule.c: In function 'k_write':
/var/lib/dkms/monmodule/0.1/build/monmodule.c:47:2: attention : format '%d'
expects argument of type 'int', but argument 2 has type 'size_t' [-Wformat]
/var/lib/dkms/monmodule/0.1/build/monmodule.c:47:2: attention : format '%d'
expects argument of type 'int', but argument 3 has type 'size_t' [-Wformat]
/var/lib/dkms/monmodule/0.1/build/monmodule.c: In function 'k_init':
/var/lib/dkms/monmodule/0.1/build/monmodule.c:88:3: attention : format '%d'
expects argument of type 'int', but argument 2 has type 'size_t' [-Wformat]
/var/lib/dkms/monmodule/0.1/build/monmodule.c:90:3: attention : format '%d'
expects argument of type 'int', but argument 2 has type 'size_t' [-Wformat]
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /var/lib/dkms/monmodule/0.1/build/monmodule.mod.o
  LD [M]  /var/lib/dkms/monmodule/0.1/build/monmodule.ko
make: quittant le répertoire " /usr/src/linux-headers-3.8.0-32-generic "
```

Le module a bien été généré et on peut visualiser sa carte d'identité avec la commande **odinfo** :

```
$ modinfo /var/lib/dkms/monmodule/0.1/`uname -r`/x86_64/module/monmodule.ko
filename:       /var/lib/dkms/monmodule/0.1/3.8.0-32-generic/x86_64/module/
monmodule.ko
license:        GPL
author:         tgayet
description:    ex-drv-char-dkms
srcversion:     3853BD18B15DD07DE2E4338
depends:
vermagic:       3.8.0-32-generic SMP mod_unload modversions
parm:           major:Static major number (none = dynamic) (int)
parm:           buf_size:Buffer size (int)
```

On peut aussi remarquer que le status du module à changé dans DKMS :

```
$ dkms status | grep monmodule
monmodule, 0.1, 3.8.0-32-generic, x86_64: built
```

Comme le module à été généré pour le noyau courant, effectuons un test de chargement rapide :

```
$ sudo insmod /var/lib/dkms/monmodule/0.1/`uname -r`/x86_64/module/monmodule.ko
$ dmng
( ... )
[261032.273926] char: allocated a 64 bytes buffer
[261032.273932] char: successfully loaded with major 250
$ lsmod | grep monmodule
monmodule              12868  0
$ rmmod ./monmodule.ko
```

8 Installation d'un module

L'installation d'un module dkms se fait là aussi en passant le paramètre **install** :

```
$ dkms install -m monmodule -v 0.1
monmodule:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
 - Installation
   - Installing to /lib/modules/3.8.0-32-generic/updates/dkms/
depmod....
DKMS: install completed.
```

Tout comme la phase d'ajout ou bien de génération, il est possible d'utiliser une version spécifique de noyau plutôt que la version courante :

```
$ sudo dkms install -m monmodule -v 0.1 -k `uname -r`
monmodule:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
 - Installation
   - Installing to /lib/modules/3.8.0-32-generic/updates/dkms/
depmod....
DKMS: install completed.
```

Le status peut être vérifié car il doit être passé à **installed** :

```
$ sudo dkms status | grep monmodule
monmodule, 0.1, 3.8.0-32-generic, x86_64: installed
```

L'installation aura installé notre module dans le répertoire **/update/dkms/** de l'arborescence lié aux modules du noyau courant :

```
$ ls -al /lib/modules/`uname -r`/updates/dkms/
total 608
drwxr-xr-x 2 root root   4096 nov.  18 10:52 .
drwxr-xr-x 3 root root   4096 oct.  22 12:00 ..
-rw-r--r-- 1 root root   8544 nov.  18 10:52 monmodule.ko
-rw-r--r-- 1 root root 504640 nov.  10 03:21 vboxdrv.ko
-rw-r--r-- 1 root root  14896 nov.  10 03:21 vboxnetadp.ko
-rw-r--r-- 1 root root  38320 nov.  10 03:21 vboxnetflt.ko
-rw-r--r-- 1 root root  36656 nov.  10 03:21 vboxpci.ko
```

Comme on le remarque, la commande **depmod** a été lancé ce qui à mis à jour le fichier de dépendances **modules.dep**.

```
$ cat  /lib/modules/`uname -r`/modules.dep | grep monmodule
updates/dkms/monmodule.ko:
```

Cela permet un chargement du module dynamiquement via la commande **modprobe** :

```
$ sudo modprobe monmodule
$ lsmod | grep monmodule
monmodule              12868  0
$ sudo  modprobe -r monmodule
```

Si jamais une installation était demandée alors que la phase de build n'était effectuée, alors dkms effectuera par dépendance d'abord la phase de build :

```
$ dkms install -m monmodule -v 0.1
Kernel preparation unnecessary for this kernel.  Skipping...
Building module:
cleaning build area....
make KERNELRELEASE=3.8.0-32-generic -C /lib/modules/3.8.0-32-generic/
build SUBDIRS=/var/lib/dkms/monmodule/0.1/build modules....
cleaning build area....
DKMS: build completed.
monmodule.ko:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
 - Installation
   - Installing to /lib/modules/3.8.0-32-generic/updates/dkms/
depmod....
DKMS: install completed.
```

9 Désinstallation

Comme nous l'aurons deviné, la désinstallation se fera sans forcément s'étendre dessus.

```
$ sudo dkms uninstall -m monmodule -v 0.1 -k `uname -r`
```

10 Suppression

La suppression du support dkms est assez similaire à l'ajout sauf qu'elle peut être retirée soit pour une version spécifique d'un noyau GNU/Linux, soit pour tous :

```
$ sudo dkms remove -m monmodule -v 0.1 --all
-------- Uninstall Beginning --------
Module:  monmodule
Version: 0.1
Kernel:  3.8.0-32-generic (x86_64)
-----------------------------------
Status: This module version was INACTIVE for this kernel.
depmod....
DKMS: uninstall completed.
---------------------------
Deleting module version: 0.1
completely from the DKMS tree.
---------------------------
Done.
```

On peut vérifier que le status du drivers est désormais inconnu par dkms car il ne doit retourner :

```
$ dkms status | grep monmodule
```

Pour la suppression pour une version donnée de noyau, le paramètre **-k** peut être là aussi utilisé :

```
$ uname -r
3.8.0-32-generic
$ sudo dkms remove -m monmodule -v 0.1 -k `uname -r`
-------- Uninstall Beginning --------
Module:  monmodule
Version: 0.1
Kernel:  3.8.0-32-generic (x86_64)
-----------------------------------
Status: Before uninstall, this module version was ACTIVE on this kernel.
monmodule.ko:
 - Uninstallation
   - Deleting from: /lib/modules/3.8.0-32-generic/updates/dkms/
 - Original module
   - No original module was found for this module on this kernel.
   - Use the dkms install command to reinstall any previous module version.
depmod....
DKMS: uninstall completed.
---------------------------
Deleting module version: 0.1
completely from the DKMS tree.
---------------------------
Done.
```

11 Utilisation avancée

11.1 Packaging DEB

Pour simplifier le travail des intégrateurs de paquets, dkms propose :

```
$ sudo dkms mkdeb -m monmodule -v 0.1 -k `uname -r`
Using /etc/dkms/template-dkms-mkdeb
copying template...
modifying debian/changelog...
modifying debian/compat...
modifying debian/control...
modifying debian/copyright...
modifying debian/dirs...
modifying debian/postinst...
modifying debian/prerm...
modifying debian/README.Debian...
modifying debian/rules...
copying legacy postinstall template...
Copying source tree...
Gathering binaries...Marking modules for 3.8.0-32-generic (x86_64) for archiving...
Creating tarball structure to specifically accomodate binaries.
Tarball location: /var/lib/dkms/monmodule/0.1/tarball//monmodule-0.1.dkms.tar.gz
DKMS: mktarball completed.
Copying DKMS tarball into DKMS tree...
Building binary package...dpkg-buildpackage:
  avertissement: utilisation d'une commande pour obtenir
  les privilèges administrateur en tant qu'administrateur
 dpkg-source --before-build monmodule-dkms-0.1
 fakeroot debian/rules clean
 debian/rules build
 fakeroot debian/rules binary
 dpkg-genchanges -b >../monmodule-dkms_0.1_amd64.changes
dpkg-genchanges: envoi d'un binaire - aucune inclusion de code source
 dpkg-source --after-build monmodule-dkms-0.1
DKMS: mkdeb completed.
Moving built files to /var/lib/dkms/monmodule/0.1/deb...
Cleaning up temporary files...
```

Le fichier .deb a bien été généré :

```
$ tree /var/lib/dkms/monmodule/0.1/deb
/var/lib/dkms/monmodule/0.1/deb
└── monmodule-dkms_0.1_all.deb

0 directories, 1 file
```

Nous pouvons lister son contenu pour vérifier :

```
$ dpkg -c /var/lib/dkms/monmodule/0.1/deb/monmodule-dkms_0.1_all.deb
drwxr-xr-x root/root         0 2013-11-18 13:59 ./
drwxr-xr-x root/root         0 2013-11-18 13:59 ./usr/
drwxr-xr-x root/root         0 2013-11-18 13:59 ./usr/src/
drw-r-xr-x root/root         0 2013-11-18 13:59 ./usr/src/monmodule-0.1/
-rw-r--r-- root/root       390 2013-11-18 11:49 ./usr/src/monmodule-0.1/dkms.conf
-rw-r--r-- root/root      2361 2013-11-18 11:49 ./usr/src/monmodule-0.1/monmodule.c
-rw-r--r-- root/root     37270 2007-03-21 05:42 ./usr/src/monmodule-0.1/dkms.spec
-rw-r--r-- root/root       207 2013-11-18 11:49 ./usr/src/monmodule-0.1/Makefile
drwxr-xr-x root/root         0 2013-11-18 13:59 ./usr/share/
drwxr-xr-x root/root         0 2013-11-18 13:59 ./usr/share/monmodule-dkms/
```

```
-rwxr-xr-x root/root      9070 2013-11-18 13:59 ./usr/share/monmodule-dkms/postinst
-rw-r--r-- root/root      3188 2013-11-18 13:59 ./usr/share/monmodule-dkms/monmodule-
0.1.dkms.tar.gz
```

11.2 Packaging RPM

Pour ce type de package, il est nécessaire de récupérer le fichier **/etc/dkms/template-dkms-mkrpm.spec** dans github (https://github.com/anbe42/dkms/blob/master/template-dkms-mkrpm.spec).

Ensuite la commande pour générer le RPM est la suivante :

```
$ sudo dkms mkrpm -m monmodule -v 0.1 -k `uname -r`
```

11.3 Déploiement sous forme d'archives

Au lieu de gérer des archives en **.deb** ou **.rpm**, il est possible de générer des tarballs des arborescences dkms :

```
$ sudo dkms mktarball -m monmodule -v 0.1 -k `uname -r`
Marking modules for 3.8.0-32-generic (x86_64) for archiving...
Marking /var/lib/dkms/monmodule/0.1/source for archiving...
Tarball location: /var/lib/dkms/monmodule/0.1/tarball//monmodule-0.1-
kernel3.8.0-32-generic-x86_64.dkms.tar.gz
DKMS: mktarball completed.
```

Le listing du **tar.gz** généré donnera :

```
$ tar -ztvf /var/lib/dkms/monmodule/0.1/tarball//monmodule-0.1-kernel3.8.0-32-
generic-x86_64.dkms.tar.gz
drwxr-xr-x root/root         0 2013-11-18 11:42 ./dkms_main_tree/
-rw-r--r-- root/root         6 2013-11-18 11:42 ./dkms_main_tree/dkms_dbversion
drwxr-xr-x root/root         0 2013-11-18 11:42 ./dkms_main_tree/3.8.0-32-
generic/
drwxr-xr-x root/root         0 2013-11-18 11:42 ./dkms_main_tree/3.8.0-32-
generic/x86_64/
drwxr-xr-x root/root         0 2013-11-18 11:42 ./dkms_main_tree/3.8.0-32-
generic/x86_64/module/
-rw-r--r-- root/root      8544 2013-11-18 11:42 ./dkms_main_tree/3.8.0-32-
generic/x86_64/module/monmodule.ko
drwxr-xr-x root/root         0 2013-11-18 11:42 ./dkms_main_tree/3.8.0-32-
generic/x86_64/log/
-rw-r--r-- root/root      1901 2013-11-18 11:42 ./dkms_main_tree/3.8.0-32-
generic/x86_64/log/make.log
drwxr-xr-x root/root         0 2013-11-18 11:42 ./dkms_source_tree/
-rw-r--r-- root/root       390 2013-11-18 11:42 ./dkms_source_tree/dkms.conf
-rw-r--r-- root/root      2361 2013-11-18 11:42 ./dkms_source_tree/monmodule.c
-rw-r--r-- root/root       207 2013-11-18 11:42 ./dkms_source_tree/Makefile
```

L'archive **tar.gz** peut être transmise sur n'importe quel système linux, quelque soit la distribution et réintégrée par la commande **ldtarball** :

```
$ dkms ldtarball -config /tmp/monmodule-0.1-kernel3.8.0-32-
generic-x86_64.dkms.tar.gz
Loading tarball for monmodule-0.1
Loading /var/lib/dkms/monmodule/0.1/3.8.0-32-generic/x86_64...
DKMS: ldtarball completed.
Creating symlink /var/lib/dkms/monmodule/0.1/source ->
                 /usr/src/monmodule-0.1
DKMS: add completed.
```

La recopie mettra le module dans l'état dkms où il était avant d'être archivé :

```
$ dkms status -k `uname -r` -m monmodule -v 0.1
monmodule, 0.1, 3.8.0-32-generic, x86_64: built
```

Le script d'init suivant peut servir pour l'installation des modules au démarrage et se fait par le paramétre **autoinstall** de la commande dkms :

```
#!/bin/sh
#
# dkms_autoinstaller        A service to automatically install DKMS
modules
#                           for new kernels.
# chkconfig: 345 04 04
# description: An autoinstaller bootup service for DKMS
#
### BEGIN INIT INFO
# Provides: dkms_autoinstaller dkms
# Default-Start: 2 3 4 5
# Default-Stop:
# Required-Start: $local_fs
# Required-Stop: $null
# Short-Description: Automatically install DKMS modules for new kernels
# Description: A service to automatically install DKMS modules for new kernels.
### END INIT INFO
test -f /usr/sbin/dkms || exit 0
#We only have these functions on debian/ubuntu
# so on other distros just stub them out
if [ -f /lib/lsb/init-functions ]; then
    . /lib/lsb/init-functions
    if [ ! -f /etc/debian_version ]; then
        alias log_daemon_msg=echo
        log_end_msg() { if [ "$1" = "0" ]; then echo " Done. "; else echo "
Failed. "; fi }
        alias log_action_begin_msg=log_daemon_msg
        alias log_action_end_msg=log_end_msg
    fi
fi
# See how we were called.
case "$1" in
  start)
    log_daemon_msg "dkms: running auto installation service for kernel $kernel"
    dkms autoinstall
    log_end_msg $?
    ;;
  stop|restart|force-reload|status|reload)
    # ignore
    ;;
  *)
    echo "Usage: $0 {start}"
esac
exit 0
```

11.4 Création d'image de drivers

Dkms permet la création d'une image de drivers souvent utile pour les installations :

```
$ dkms mkdriverdisk -d redhat -m monmodule -v 0.1 -k `uname -r`
Creating Red Hat v1 driver disk.
Module monmodule/0.1 already built for kernel 3.8.0-32-generic/4
Marking 3.8.0-32-generic/x86_64/module/monmodule.ko...
: File: modinfo not found in /var/lib/dkms/monmodule/0.1/source/redhat_driver_disk/
: File: disk-info not found in /var/lib/dkms/monmodule/0.1/source/redhat_driver_disk/
: File: modules.dep not found in /var/lib/dkms/monmodule/0.1/source/redhat_driver_disk/
: File: pcitable not found in /var/lib/dkms/monmodule/0.1/source/redhat_driver_disk/
: File: modules.pcimap not found in /var/lib/dkms/monmodule/0.1/source/redhat_driver_disk/
: File: pci.ids not found in /var/lib/dkms/monmodule/0.1/source/redhat_driver_disk/
Creating driver disk on floppy media:
compressing modules.cgz....
Copying files /tmp/dkms.dsV1RL
making a blank floppy image....
mkdosfs....
loopback mounting disk image...
  copying files to floppy disk image...
unmounting disk image...démontage : /tmp/dkms.qn80Q1 : périphérique occupé.
```

```
      (Dans certains cas, des infos sur les processus l'utilisant
       sont récupérables par lsof(8) ou fuser(1))
(bad exit status: 1)
rm: impossible de supprimer "/tmp/dkms.qn80Q1": Périphérique ou ressource occupé
Disk image location: /var/lib/dkms/monmodule/0.1/driver_disk/monmodule-0.1-
kernel3.8.0-32-generic-x86_64-dd.img
DKMS: mkdriverdisk completed.
```

12 Limitations

Bien que aujourd'hui bien « huilé », dkms possède certaines limitations :

- dkms ne fonctionne pas sur toutes les versions de noyau,
- le système doit disposer d'un toolchain (compilateur **gcc**, editeur de lien **ld**, ...),
- les headers noyau doivent être installés,
- sur de faibles configurations matérielles, le temps de regénération des drivers peut être important à chaque modification de la version du noyau.
- Les modules ne seront pas chargés s'ils sont dans initrd.

Conclusion

Avec dkms, vous voilà à l'abris d'un gros problème de version de drivers suite à une mise à jour de votre noyau ou des sources d'un driver. Dkms est un système offrant l'assurance de garder la continuité de service. ■

Liens

- http://linux.dell.com/dkms/dkms.html
- http://linux.dell.com/dkms/manpage.html
- http://linux.dell.com/git/?p=dkms.git
- http://www.dell.com/downloads/global/power/1q04-ler.pdf
- http://linux.dell.com/dkms/dkms-ols2004.pdf
- http://wiki.mandriva.com/fr/DKMS
- https://help.ubuntu.com/community/RocketRaid
- https://help.ubuntu.com/community/DKMS
- http://fasmz.org/~pterjan/doc/dkms.html
- https://wiki.archlinux.org/index.php/Dynamic_Kernel_Module_Support
- https://xkyle.com/building-linux-packages-for-kernel-drivers/
- https://help.ubuntu.com/community/Kernel/DkmsDriverPackage
- http://tomoconnor.eu/blogish/building-dkms-package-latest-intel-e1000e-driver/#.UTxWEjJySyg
- http://www.linuxjournal.com/article/6896
- http://www.lea-linux.org/documentations/HOWTO_Dkms
- http://askubuntu.com/questions/53364/command-to-rebuild-all-dkms-modules-for-all-installed-kernels
- https://github.com/anbe42/dkms/blob/master/template-dkms-mkrpm.spec

CORRECTION GÉOMÉTRIQUE D'IMAGES PRISES EN VUE OBLIQUE

PROJECTION SUR MODÈLE NUMÉRIQUE D'ÉLÉVATION POUR EXPLOITATION QUANTITATIVE DE PHOTOGRAPHIES NUMÉRIQUES

par J.-M Friedt

La gestion d'informations spatialisées devient accessible au grand publique avec la prolifération des récepteurs GPS dans les téléphones portables et les appareils photographiques numériques. Dans cette présentation, nous nous proposons d'exploiter quantitativement les photographies numériques prises en vue oblique en les projetant en vue azimutale par déformation géométrique s'appuyant sur des points de contrôle géo-référencés et de draper un modèle numérique d'élévation pour ajouter une troisième dimension aux analyses. Bien que recourant à des outils au travers d'une interface graphique, nous nous efforçons de conserver la possibilité d'appliquer les opérations en ligne de commande afin de pouvoir scripter les traitements sur un grand nombre d'images et ainsi traiter des séries temporelles telles que fournies par une webcam observant un glacier alpin pour en évaluer la couverture neigeuse.

1 Introduction

La gestion spatiale de l'information est accessible à tous depuis la prolifération des récepteurs GPS permettant de géolocaliser à peu près toute activité (par exemple, application OSMTracker[1] pour Android).

De plus, de nombreuses opportunités se présentent avec des images de paysages prises en vue oblique **[1]**, par exemple des photographies numériques acquises depuis des points élevés ou d'avion. Nous désirons fusionner ces deux sources de données que sont les évènements géolocalisés, les informations cartographiques et ce, complétées par des photographies prises en vue oblique par un observateur au sol afin de palier à la rareté et à la difficulté d'obtentions d'images aériennes. Notons cependant que les méthodes de traitement du signal proposées ici sont adéquates pour l'exploitation d'images prises depuis des drones personnels tels que la pléthore d'hélicoptères ou quadricoptères qui pullulent actuellement, munis d'instruments de prises de vues (webcams, appareils photographiques numériques).

La déformation géométrique entre le premier plan et l'arrière plan rend l'analyse quantitative **[2]** des informations complexes, en particulier en présence d'un terrain qui n'est pas plat. Ce problème classique dans le cas des prises de vues aériennes et satellitaires

[1] http://code.google.com/p/osmtracker-android/

en orientation quasi-azimutale (angle par rapport à la normale du sol faible) est résolu par une approche rigoureuse de projection des images sur un modèle numérique de terrain ne nécessitant que les caractéristiques de l'instrument de prise de vue, sa position et son orientation au moment de la capture d'image **[3]**. Cette méthode se révèle cependant inexploitable dans le cas de prises de vues par appareils photographiques numériques ou webcams depuis le sol du fait de l'angle trop important par rapport à la normale au sol. Par ailleurs, les caractéristiques optiques de l'instrument de prise de vue sont souvent insuffisamment précises pour les exploiter directement, bien que certains auteurs aient sélectionné cette approche, par exemple nécessaire pour l'analyse stéréographique **[4]**.

Une approche alternative consiste donc à se reposer sur un certain nombre de points de référence au sol - nommés Ground Control Points (GCP) – pour effectuer une bijection entre coordonnées sur une image et position sur le terrain. Cette approche ne repose sur aucune connaissance physique de l'instrument de prise de vue et se contente d'établir la relation locale entre des points dans l'espace et un pixel. Divers modèles d'interpolation sont proposés en fonction des connaissances de l'utilisateur sur l'environnement étudié. Finalement, le modèle numérique d'élévation (MNE) est utile pour projeter la photographie ainsi déformée pour un rendu aussi réaliste que possible et valider la cohérence avec la topographie.

Deux approches opposées pour l'analyse quantitative des informations contenues dans les photographies prises en vue oblique consistent donc soit à identifier la surface associée à chaque pixel en déformant une carte pour correspondre à la prise de vue[2] ou, au contraire, à déformer l'image prise en vue oblique pour correspondre à la vue qui aurait été obtenue en projection azimutale **[5]**. Cette seconde méthode sera préférée car elle permet de fusionner des sources diverses dans un référentiel commun.

Il semble donc évident que plus le nombre de GCPs est important, plus la déformation géométrique de l'image épousera au mieux le modèle numérique d'élévation en tenant compte de variations locales de topographie. Cependant, la capture de GCPs – par exemple au moyen d'un récepteur GPS en se positionnant sur des points marquants de la photographie – est une activité fastidieuse et gourmande en temps, voir complexe si la cible remarquable a le mauvais goût d'être mobile. Nous proposerons donc l'alternative de se reposer sur Google Earth pour identifier les coordonnées GPS (latitude, longitude) de quelques pixels remarquables sur la photographie, puis de s'appuyer sur ces points pour déformer de façon appropriée l'image pour recouvrir un modèle numérique d'élévation.

Un certain nombre d'outils propriétaires existent pour effectuer ces opérations **[6]**. Notre objectif dans cette présentation tient en deux points : d'une part utiliser exclusivement du logiciel libre pour atteindre notre objectif et d'autre part être capable de scripter le traitement de séries de photographies, toutes prises du même site et selon la même orientation, afin d'automatiser la procédure de génération des images en projection azimutale. Le second soucis tient au désir d'utiliser des séquences de captures périodiques, par exemple grâce aux webcams diffusant continuellement des images sur le web, afin d'effectuer un traitement quantitatif d'analyse des images. Seule une approche exploitant la ligne de commande permet d'envisager un traitement en série d'une masse importante d'images. Dans tous les cas, une image aérienne ou satellitaire supportera l'identification des points remarquables entre la photographie analysée et le terrain, et fournira une référence sur laquelle se caler. Finalement, nous voudrons projeter en 3-dimensions le résultat du calcul pour en valider la pertinence.

2 QGis et accès à GRASS

Deux outils se démarquent dans la panoplie des logiciels libres disponibles pour le traitement d'informations géographiques compatibles avec une utilisation en ligne de commande pour le traitement séquentiel d'un grand nombre de fichiers de données : GRASS (Geographic Resources Analysis Support System, http://grass.osgeo.org/) et la bibliothèque GDAL (Geospatial Data Abstraction Library, http://www.gdal.org/). Ces deux outils, d'une prise en main un peu ardue, sont intégrés comme greffons (plugins) de l'interface graphique QGis. Nous proposons donc la séquence de développements suivante :

1. exploitation de QGis pour les opérations les plus simples telles que charger des séries de points d'élévations (MNE – Digital Elevation Model ou DEM en anglais[3]) ou images,
2. exploitation de QGis et de son greffon Georeferencer pour afficher une image prise en vue oblique et association de coordonnées GPS (points de contrôle – GCP) à des pixels remarquables de l'image. Ces opérations sont manuelles et donc difficiles à étendre à un jeu de données important,
3. exploitation de GDAL dont la ligne de commande a été générée par QGis pour réaliser l'opération de correction géométrique depuis le shell,
4. pour le moment, GRASS ne sert que depuis QGis pour quelques traitements de données additionnelles (affichage en 3D du MNE

[2] L'auteur a vu cette méthode mise en œuvre sur des clichés obtenus par photographie analogique, avant la prolifération du traitement numérique d'images, en plaquant un calque contenant des polygones de surface constante sur une image prise depuis le sol. Compter le nombre de polygones recouvrant une surface de couleur claire dans la photographie permettait d'estimer alors la couverture neigeuse sur un massif alpin.

[3] http://eductice.ens-lyon.fr/EducTice/recherche/geomatique/veille/imagerie-3d/MNT

drapé, affichage des triangles de Delaunay) mais il est envisageable que les traitements effectués par GDAL y soient aussi accessibles, l'inspiration initiale de ce document provenant de http://grass.osgeo.org/wiki/Orthorectification_digital_camera. Cependant l'approche proposée consistant à connaître aussi bien que possible les conditions de prises de vues (position, ouverture angulaire, aberrations optiques) ne sont pas compatibles avec notre approche de déformation de l'image selon des points de contrôle, qui elle semble accessible par ailleurs (commande **i.rectify**).

La bonne pratique voudrait que nous partions d'images non géo-référencées, que nous allions obtenir manuellement la position spatiale de points de référence au sol (GCP) et que nous exploitions ces informations pour déformer les images en accord avec divers algorithmes d'étirement d'une feuille souple (rubber sheeting). En pratique, nous ne démontrerons qu'une fois l'acquisition manuelle de points GPS et exploiterons les coordonnées GPS en référentiel WGS84 fournies par Google Earth dans la suite du document.

3 Sources de données

Nous avions présenté dans ces pages une exploitation de trames GPS dans divers environnements de traitement, dont GRASS **[7]**. Les outils ont passablement évolué depuis la rédaction de cette prose, avec l'apparition de la version stable (1.x) de Quantum GIS (nommé dans ce document QGis tel que indiqué à http://www.qgis.org/) autour de janvier 2009. Cet environnement graphique de gestion d'informations géographiques facilite la prise en main de divers outils au travers de greffons (plugins) qui complètent des fonctions de base aussi simples que le chargement d'informations matricielles (raster tel qu'un modèle numérique de terrain où chaque pixel se voit assigné une altitude) ou vectorielles (points GPS par exemple). Lorsque toutes ces informations sont géo-référencées dans un repère et un mode de projection commun, leur fusion se fait naturellement (Fig. 1). La compilation de l'ensemble des outils – graphiques et bibliothèques associées – est quelque peu fastidieuse et nous nous contentons d'exploiter les paquets pré-compilés pour plate-forme x86 distribués par QGis à http://www.qgis.org/en/site/forusers/alldownloads.html#debian. En particulier nous installons **qgis**, **qgis-plugin-grass** et, depuis les paquets Debian, **gdal-bin** ainsi que les dépendances associées.

Dans l'exemple qui va suivre, nous nous sommes contentés de demander à QGIS d'importer un fichier de données vectorielles contenant une trace GPS stockée au format KML (automatiquement reconnu) et un modèle numérique de terrain au format GeoTIF, à savoir un fichier bitmap contenant des méta-données sur la taille du pixel et la coordonnée géographique d'un coin.

Nous exploitons 3 sources de données librement disponibles :

1. un modèle numérique de terrain parmi tous ceux actuellement disponibles depuis divers sites et notamment de la NASA (http://grass.osgeo.org/wiki/Global_datasets et http://asterweb.jpl.nasa.gov/gdem.asp). Nous avons pour notre part obtenu les images ASTER GDEM2 depuis http://earthexplorer.usgs.gov/ ou https://reverb.echo.nasa.gov. La résolution spatiale est de une seconde d'arc soit environ 30 m en latitude et 15 m en longitude pour la France. Il est cependant apparu à l'usage que GDEM2 est encore affecté d'un certain nombre d'incohérences ou de zones d'altitude indéfinie qui nécessitent une correction manuelle, faute de quoi le rendu des calculs n'est pas esthétiquement acceptable **[8]**. Nous nous sommes donc par la suite contentés du modèle numérique de terrain SRTM obtenu depuis la navette spatiale et qui couvre le monde avec une résolution de 100 m à l'équateur (3 secondes d'arc corrigé, disponible à http://srtm.csi.cgiar.org/ et en particulier http://srtm.csi.cgiar.org/SELECTION/inputCoord.asp),
2. des images satellitaires telles que issues de la base de données Landsat ou, présentant une meilleure résolution spatiale et un meilleur suivi temporel, les previews des images haute résolution commercialisées par DigitalGlobe et accessibles gratuitement depuis https://browse.digitalglobe.com,
3. des images en vue oblique soit obtenues manuellement par l'auteur, soit disponibles en licence CC sur internet (par exemple panoramio.com),
4. le positionnement des points de référence au sol au moyen de Google Earth qui fournit la position GPS avec une précision inférieure à la taille du pixel des MNEs ou des images satellites qui nous concernent ici. Une source **[9]** annonce que les divers globes virtuels numériques sont cohérents entre eux à mieux que ±10 cm.

Le MNE est chargé sous forme de fichier GeoTiff, un fichier bitmap contenant un certain nombre de méta-données en complément de la matrice des altitudes, depuis le site http://earthexplorer.usgs.gov/ ou http://gdex.cr.usgs.gov/gdex/ (nécessite un enregistrement). Nous constatons par **gdalinfo** - outil lié à la bibliothèque GDAL - que

```
$ gdalinfo 201210280425O8_188302717.tif
Driver: GTiff/GeoTIFF
Files: 201210280425O8_188302717.tif
Size is 5260, 3342
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
Origin = (5.454712000000000,47.672423999999999)
```

```
Pixel Size = (0.000277791254753,-0.000277780969479)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (   5.4547120,  47.6724240) (  5d27'16.96"E, 47d40'20.73"N)
Lower Left  (   5.4547120,  46.7440800) (  5d27'16.96"E, 46d44'38.69"N)
Upper Right (   6.9158940,  47.6724240) (  6d54'57.22"E, 47d40'20.73"N)
Lower Right (   6.9158940,  46.7440800) (  6d54'57.22"E, 46d44'38.69"N)
Center      (   6.1853030,  47.2082520) (  6d11' 7.09"E, 47d12'29.71"N)
```

le système de coordonnées (WGS84) ainsi que les coordonnées des quatre coins sont renseignés dans divers champs du fichier TIF.

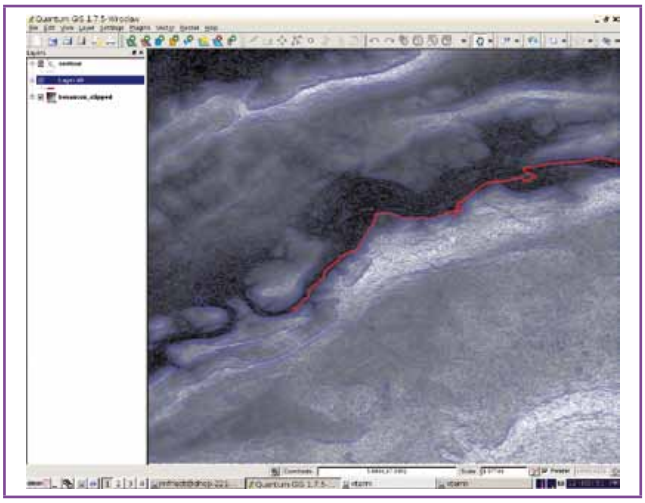


Figure 1 : Fusion de données matricielles (raster) de modèle numérique d'élévation (MNE) et de données vectorielles d'un trajet acquis par GPS au rythme de 1 point/s. Le MNE a été complété des lignes de niveau.

Ce fichier est chargé dans QGis au moyen de ***Layer > Add raster layer***. Diverses analyses triviales sont possibles et notamment après ajout du greffon GDALTools, telles que la génération des courbes de niveau[4] (Fig. 1) par ***Raster > Extraction > Contour*** qui fournit par ailleurs la commande GDAL suivante :

```
gdal_contour -i 10.0 /home/jmfriedt/grass/20121028042508_188302717.tif
```

De la même façon, QGis est capable de charger un fichier KML contenant les coordonnées GPS des points successifs d'un trajet, qui se superposent donc aux données précédentes sous réserve évidemment de se placer dans le même référentiel géographique (ici WGS84, référentiel par défaut du GPS).

4 Géoréférencement d'images

Il est difficile de se convaincre sur un MNE de la validité du positionnement. Nous allons donc nous efforcer d'ajouter à cette information matricielle d'altitude une image aérienne.

Nous avons déjà décrit comment Google Maps est une source possible, à défaut d'être légale, d'obtenir des images géo-référencées avec une excellente résolution **[10]**. Nous nous contentons ici de reprendre une image de preview de DigitalGlobe[5] **[11]**. Le module **georeferencer** de QGis (Fig. 4) propose l'environnement nécessaire pour placer les points de contrôle au sol (GCP) et relier une position sur l'image à une information spatiale bi-dimensionnelle (indépendamment de toute information d'altitude qui n'est pas nécessaire lors de la déformation de l'image).

4.1 Ajout d'une image satellitaire

Nous avons choisi d'exploiter l'image satellite obtenue lors du passage au-dessus de Besançon acquise le 26 Avril 2011 et disponible en preview sur le site de DigitalGlobe. Compte

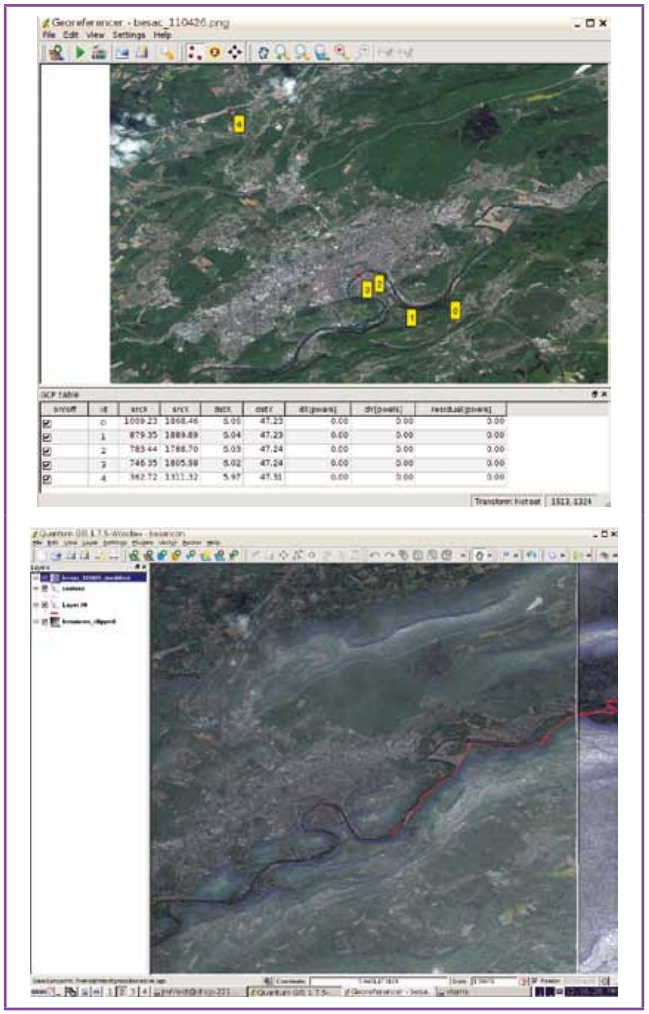


Figure 2 : Haut : géo-référencement d'une image satellite obtenue sur DigitalGlobe au moyen de quelques points GPS de zones remarquables identifiées sur Google Earth. Bas : superposition de l'image satellite géo-référencée, le MNE et la trace GPS pour valider le positionnement de la piste cyclable le long du Doubs.

[4] http://nathanw.net/2011/09/27/generating-contours-using-gdal-via-shell-or-qgis/

[5] https://browse.digitalglobe.com

tenu de la trajectoire des satellites en orbite basse (LEO), les images sont toujours orientées selon un axe nord-sud et les motifs y sont aisément repérables (Fig. 2).

Une fois le greffon **georeferencer** de QGis lancé, nous chargeons l'image obtenue sur DigitalGlobe et sélectionnons quelques points remarquables pour la positionner dans l'espace. Nous faisons l'hypothèse que DigitalGlobe a déjà effectué le travail de projection de l'image en vue azimutale et qu'il suffit d'une translation et d'une homothétie, voir éventuellement une rotation, pour placer tous les points de l'image dans l'espace. Une interpolation linéaire sur peu de points de contrôle dans les coins de la zone d'analyse qui nous concerne semble donc adéquate.

4.2 Positionnement d'une carte

Avant de nous engager dans l'étape plus avancée de correction géométrique d'image en prise oblique, attelons nous à résoudre un cas simple, à savoir géo-référencer une image en prise de vue azimutale non-référencée (image issue des previews de DigitalGlobe sur la région parisienne – dans notre cas nous avons pris la version la mieux résolue de la bande acquise le 3 juillet 2011) et y superposer une carte non géo-référencée et sans vocation initiale à l'être, à savoir le plan du métro parisien. Cet exemple est sélectionné par la multitude de GCPs évidents à identifier sur Google Earth, sur l'image satellitaire (intersections de boulevards, parcs clairement visibles depuis l'espace) et sur la carte du métro.

Un résumé de la séquence d'analyse consiste donc en :

1. charger l'image dans le greffon **georeferencer** et identifier la position GPS (latitude/ longitude) pour suffisamment de pixels (X,Y) pour permettre d'identifier le modèle numérique de terrain, soit en retrouvant les coordonnées dans Google Earth, soit en identifiant sur une image déjà géoréférencée avec suffisamment de précision,
2. charger dans QGis au moyen de l'icone d'ajout de fichier matriciel (*raster*),
3. charger dans GRASS à partir de QGis au moyen de **r.in.gdal.qgis**,
4. ajouter dans GRASS au moyen de l'icône d'ajout d'un fichier raster,
5. ajuster à la taille de la région d'intérêt de GRASS au moyen de **r.resample**,
6. dans cet exemple, cette procédure est réitérée d'abord en repérant les coordonnées sur Google Earth pour l'image satellite, puis en repérant les coordonnées des stations de métro sur l'image ainsi positionnée.

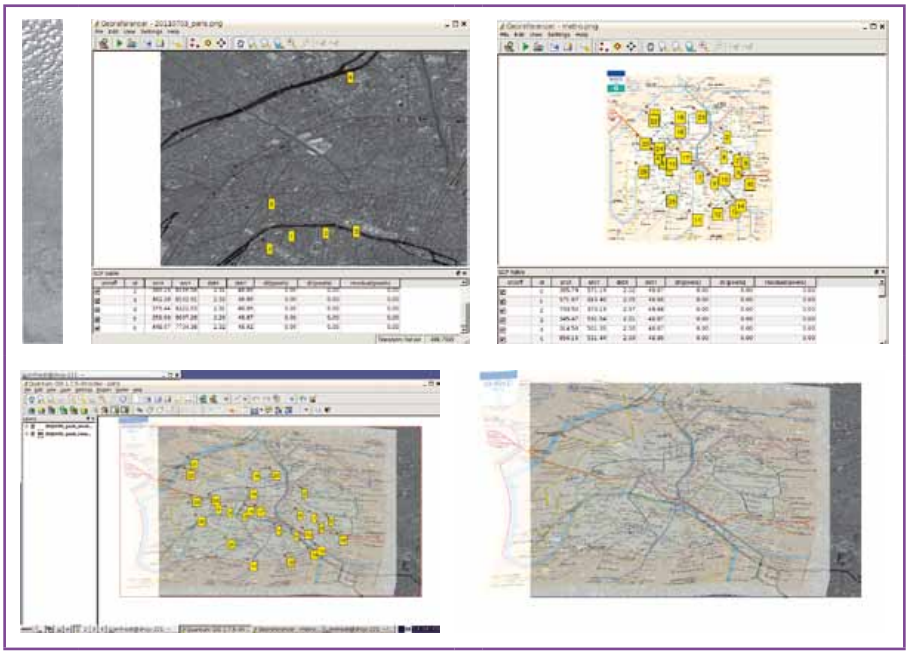


Figure 3 : De gauche à droite : une bande acquise par satellite et disponible en preview sur le site de DigitalGlobe, le géo-référencement de l'image satellite, le géo-référencement d'une carte du métro parisien en sélectionnant des points remarquables pour quelques stations et, finalement, déformation de la carte de métro pour tenir compte de la position géographique des stations sélectionnées.

Une fois les points de contrôle sélectionnés, le choix du mode de déformation et d'interpolation doit être sélectionné (symbole de la clé, ou ***Settings > Transformation settings***) : nous sélectionnons en général ***Thin Plate Spline*** pour la transformation et les voisins les plus proches (*Nearest neighbour*) pour l'interpolation. Il est judicieux de ne pas activer l'option de définition de la résolution cible (***Set Target Resolution***) qui ne permet pas de trouver de solution. Un exemple de résultat de ce traitement est proposé en Fig. 3.

4.3 Cas de l'image en prise de vue oblique

Nous nous proposons de réitérer la procédure de correction géométrique d'une photographie prise en vue oblique – ici depuis un ballon captif – pour la géoréférencer et ainsi compléter une image en vue azimutale (Fig. 4).

Le fichier de GCP est en ASCII (Fig. 5) avec une première ligne d'entête inutile : ce fichier est donc trivialement exploitable pour fabriquer la ligne de commande exploitant la bibliothèque GDAL afin d'incorporer ces points de référence dans le fichier GeoTIFF ou, par ailleurs, avec un peu de modification (élimination de la première ligne) chargement par GRASS.

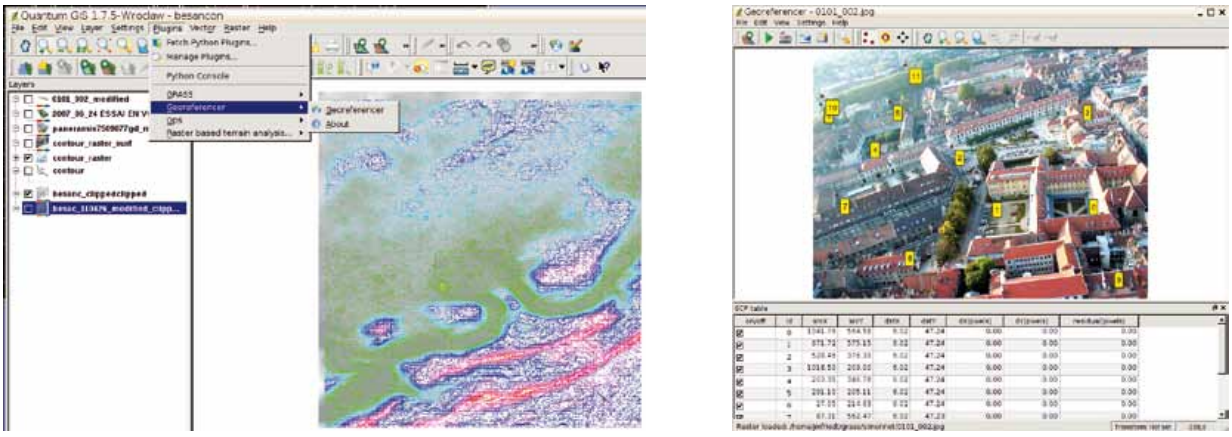


Figure 4 : Gauche : lancement du greffon georeferencer de QGis, une interface graphique pour facilement associer une position GPS à un pixel d'une image. Droite : georeferencer combine une interface représentant l'image bitmap à corriger et les points de contrôle au sol définis dans l'interface graphique au moyen de la souris. Néanmoins, l'imprécision de l'entrée par l'interface graphique est aisément corrigée en modifiant le fichier ASCII contenant les points (Fig. 5).

L'outil pour incorporer les GCP dans le fichier GeoTIFF est **gdal_translate** (traduction d'un format d'image vers un autre – ici de JPEG à GeoTIFF) : QGis nous propose par ailleurs de fabriquer cette ligne de commande pour nous. Une fois le fichier GeoTIFF convenablement renseigné, la commande **gdalwarp** effectue la correction géométrique elle-même selon une loi d'interpolation définie par l'utilisateur.

```
/usr/bin/gdal_translate -of GTiff \
-gcp 1041.76 564.581 6.02363 47.2353 \
-gcp 671.718 575.154 6.02292 47.235  \
-gcp 528.458 376.388 6.02153 47.2354 \
-gcp 1018.5 202.996 6.02219 47.2367  \
-gcp 203.348 346.784 6.01993 47.2351 \
-gcp 291.101 205.11 6.01816 47.236   \
-gcp 27.0485 214.626 6.01583 47.2355 \
-gcp 87.3128 562.467 6.02131 47.2344 \
-gcp 335.771 761.233 6.0228 47.2345  \
-gcp 1138.24 843.7 6.02419 47.2349   \
-gcp 32.0705 182.907 6.01508 47.2357 \
-gcp 350.837 63.4361 6.01447 47.238  \
"0101_002.jpg" "/tmp/0101_002.jpg"

/usr/bin/gdalwarp -r near -tps -co COMPRESS=NONE  "/tmp/0101_002.jpg"
"0101_002_jmfout.tif"
```

Seule la dernière étape du calcul mérite d'être scriptée pour un traitement automatique, l'identification des GCPs étant nécessairement une analyse manuelle fastidieuse mais qui ne peut être automatisée.

Nous avons constaté qu'une difficulté majeure de localisation de pixels d'images acquises en environnement urbain est la difficulté à visualiser la base des bâtiments ou, de façon générale, des points de référence au niveau du sol (Fig. 6). L'estimation de la qualité de la correction géométrique semble donc en l'état complexe, mais reste là encore un problème ouvert qu'il faudra résoudre pour généraliser la méthode de

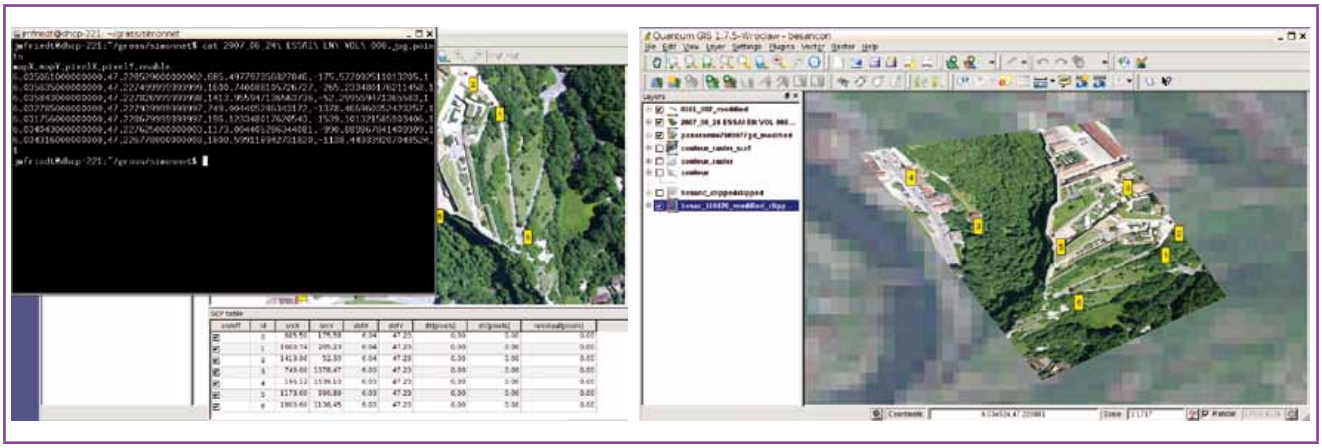


Figure 5 : Gauche : points de contrôle au sol sélectionnés graphiquement (en arrière plan) et, en haut à gauche de la figure, le fichier ASCII ainsi généré. Droite : projection de la photographie sur le fond d'image satellitaire illustrant le gain en résolution, le choix des points de contrôle et une certaine cohérence dans la continuité des motifs visibles sur les deux images.

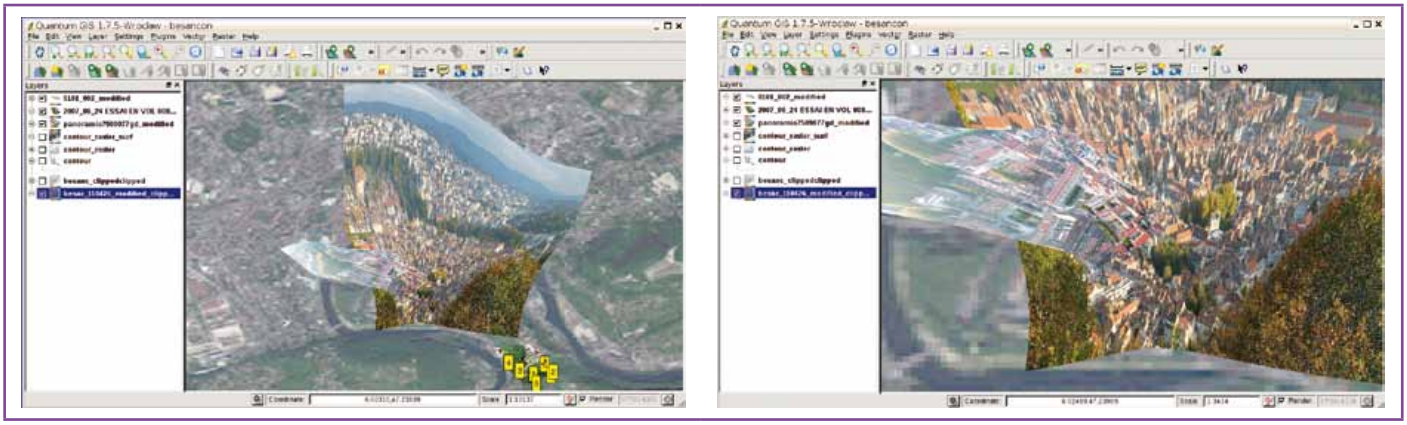


Figure 6 : Diverses images prises dans des conditions plus ou moins favorables (angle rasant sur des terrains présentant un dénivelé important) sont comparées après corrections géométriques. Droite : gros plan sur la zone d'intersection de deux des images. L'estimation de la qualité de la correction est rendue complexe par l'absence de visibilité de points de référence au sol, cachés par les bâtiments dans ces vues obliques.

traitement (Figs. 7 et 9). Dans ce contexte, la photographie aérienne prise au dessus du terrain relativement dégagé de la citadelle de Besançon (Fig. 5, droite) semble un cas idéal malgré l'angle important des pentes autour de la forteresse qui induit des déformations significatives et donc la nécessité de sélectionner de nombreux points de contrôle.

5 De QGis à GRASS : visualisation 3D

Un point quelque-peu déroutant initialement est que bien que QGis fasse appel aux fonctions de GRASS au travers du greffon approprié, les ensembles de données gérées par les deux outils sont distincts. Ce n'est donc *pas* parce-que QGis connaît l'existence d'une couche qu'elle peut être traitée par GRASS. Il faut au préalable créer une carte GRASS et y incorporer les couches matricielles et vectorielles de QGis qui doivent y être traitées.

La principale motivation d'exploiter GRASS dans ce contexte tient en l'outil de visualisation 3D permettant de draper une image sur un modèle numérique de terrain et observer le résultat comme une surface en 3D. La procédure est décrite en détail à http://linfiniti.com/2010/12/3d-visualisation-and-dem-creation-in-qgis-with-the-grass-plugin/ mais le résultat est quelque-peu décevant (Fig. 8) en comparaison de l'outil professionnel de ArcGIS que sont 3D Analyst et ArcScene, en particulier lorsque le MNE GDEM2 est utilisé du fait d'un bruit excessif sur le jeu de données. Le résultat reste néanmoins exploitable pour valider la projection (Fig. 9). Le MNE (GDEM2) de ASTER est trop inhomogène pour permettre une projection propre. Nous avons constaté que malgré une résolution théorique annoncée inférieure, utiliser un raster de SRTM, plus homogène, fournit un résultat visuellement meilleur. L'alternative consistant à lisser GDEM2 par convolution donnera probablement la même résolution que SRTM et mieux vaut donc s'appuyer sur un jeu de données propre et validé.

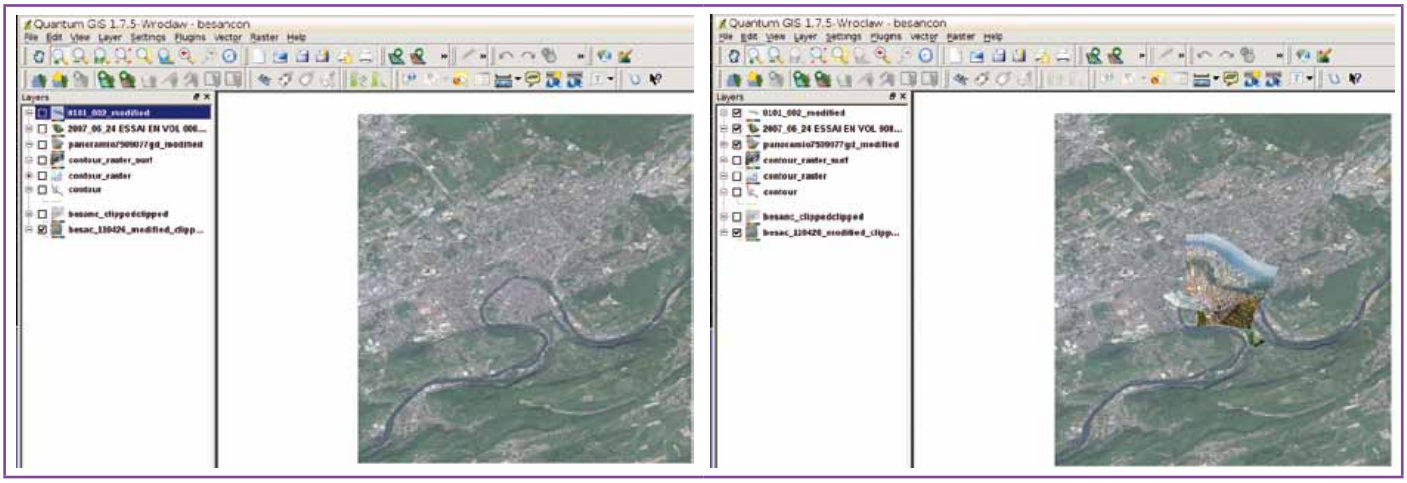


Figure 7 : Comparaison de l'image de fond corrigée géométriquement pour que quelques points de référence soient correctement positionnés (image satellitaire donc déjà corrigée pour tenir compte du relief du terrain, gauche), et ajout des photographies en prise obliques (droite – photographie disponible sur Panoramio sous l'identifiant 7509077).

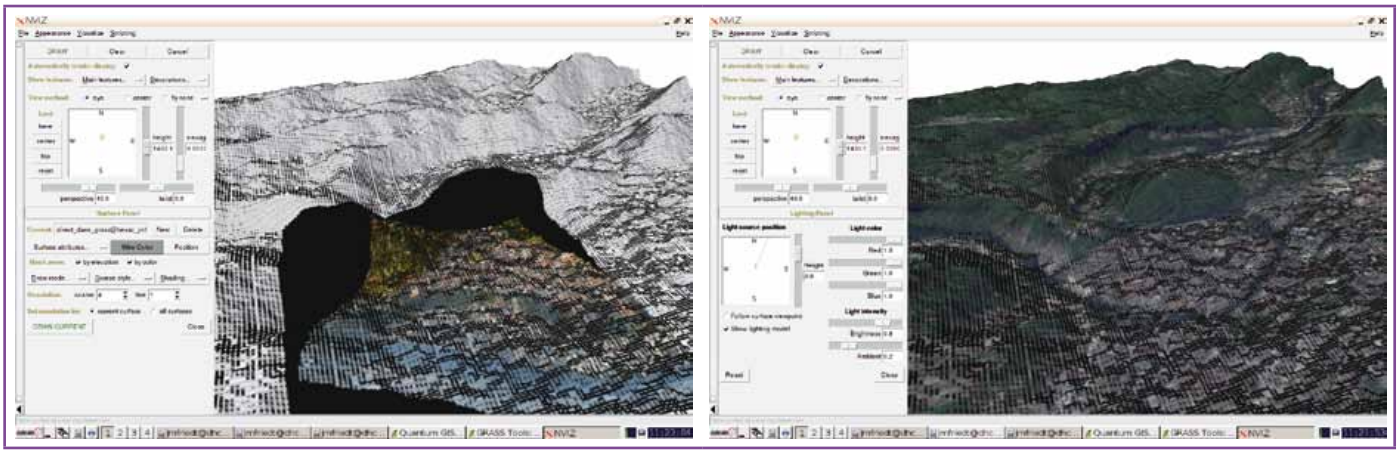


Figure 8 : Drapage de diverses images (prise de vue oblique ou azimutale par satellite) sur un modèle numérique d'élévation (MNE). Noter la cohérence de la position du Doubs qui, après avoir formé la boucle autour de Besançon, suit bien un tracé dans les vallées adjacentes.

Néanmoins, la disponibilité des outils classiques de traitement et d'analyse des informations spatiales semble justifier la compatibilité des méthodes de traitement avec GRASS, en se remémorant que le travail de correction géométrique et mosaïqué des images n'est que l'étape préliminaire au traitement quantitatif des informations contenues dans les vues azimutales (Fig. 10). Les triangles de Delaunay affichés sur cette figure représentent les surfaces formées par les plus proches voisins entre lesquels une interpolation sera faite lors de la correction géométrique des images – le choix de la triangulation vérifie qu'aucun autre GCP que les 3 sommets du triangle n'est inclus dans le cercle passant par les 3 sommets de chaque triangle. Les polygones de Voronoï[6] (en vert sur Fig. 10) partitionnent l'espace en sphères d'influence : chaque polygone est centré sur un GCP (point connu) et s'étend jusqu'à ce que l'influence d'un autre point de référence devienne dominant. Les triangles de Delaunay et polygones de Voronoï sont une représentation duale du partitionnement de l'espace : les sommets des polygones de Voronoï sont les centres des cercles s'appuyant sur les sommets des triangles de Delaunay et chaque sommet d'un triangle de Delaunay est relié au polygone de Voronoï voisin. Il s'agit donc de distributions rationnelles de l'espace en régions d'influence de points connus permettant d'interpoler au mieux les grandeurs entre ces points de référence.

Maintenant qu'une image en prise de vue oblique est représentée en vue azimutale, avec chaque pixel couvrant une même surface sur l'image déformée, nous pouvons envisager d'analyser quantitativement les informations contenues dans les paysages ainsi observés.

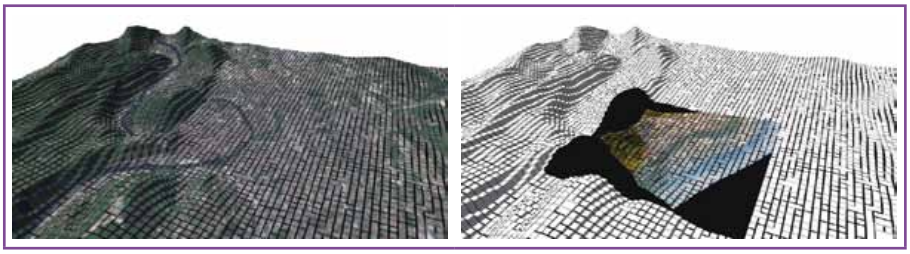


Figure 9 : Drapage de diverses images (prise de vue oblique ou azimutale pas satellite) sur un modèle numérique d'élévation (MNE). Noter la cohérence entre la végétation sur la vue oblique et le MNE : la forêt cesse en bas des côtes pour laisser place à l'urbanisation.

6 Tout ça pour ça ... application au glacier des Pélerins

Le glacier des Pélerins **[12, p.116]** est un petit glacier au pied de l'Aiguille du Midi dans la vallée de Chamonix. Il s'avère être au premier plan de la vue de la webcam placée au sommet de l'Aiguille du Midi par la Compagnie du Mont Blanc[7]. En l'absence d'une présence physique sur place, ces images capturées automatiquement 6 fois par jours à intervalle de temps de 2 h par un `crontab` fournissent la base de données nécessaire à l'étude de la couverture neigeuse **[13]** de ce petit glacier de 1,2 km^2 de superficie s'étendant jusqu`a une altitude de 2350 m **[14, p.172]**.

Nous avons dans un premier temps obtenu la section du MNE SRTM pour la région contenant le massif du Mont Blanc, ainsi que l'image en tons de gris DigitalGlobe d'identifiant 1020010008CEF300 acquise le 24 juin 2009 (accessible depuis l'onglet Catalog de https://browse.digitalglobe.com). La figure 11 illustre le drapage sur le MNE de cette image satellitaire positionnée au moyen de

[6] http://mathworld.wolfram.com/VoronoiDiagram.html

[7] http://www.compagniedumontblanc.fr/fr/webcams et en particulier http://www.compagniedumontblanc.fr/webcam/Aiguille.jpg

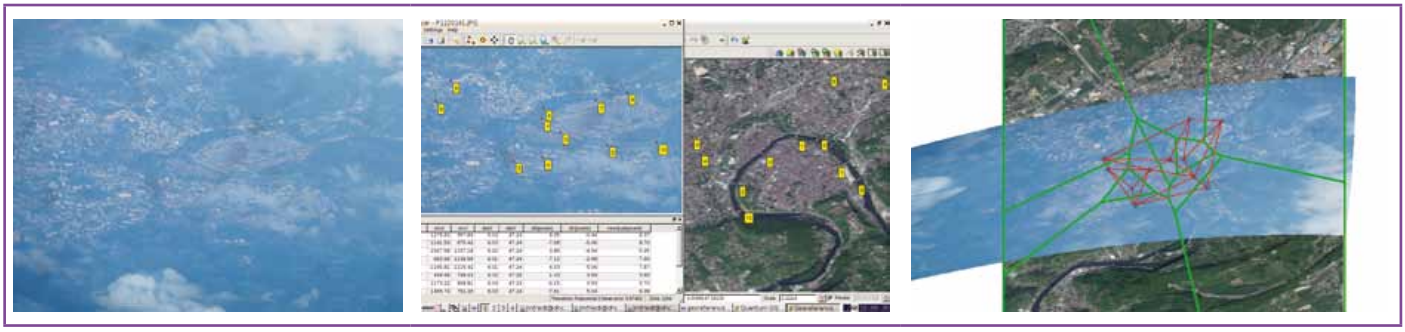


Figure 10 : Au delà de la correction géométrique d'images – ici selon des polynômes de degré 2 – et la visualisation en 3D du drapage de l'image sur le MNE, l'exploitation des données dans GRASS donne accès aux outils associés tels que l'affichage des triangles de Delaunay ou des polygones de Voronoï correspondant. Gauche : photographie prise par l'auteur depuis un vol Heraklion-Paris qui passe à proximité de Besançon. Noter les ponts sur la boucle du Doubs qui serviront de GCP lors de la correction géométrique de l'image (milieu). Droite : en bleu les GCPs, en vert les polygones de Voronoï, en rouge les triangles de Delaunay.

quelques GCP sélectionnés autour du Mont Blanc, ainsi qu'un échantillon des photographies de webcam que nous allons exploiter au cours de corrections géométriques par l'identification, dans Google Earth, de points remarquables. Le village de Chamonix est visible au fond de la vallée et servira de référence lors de la recherche des images floues (mauvais temps et webcam dans les nuages). Une mise en contexte, résultat du traitement qui va être décrit ci-dessous, est proposé sur la Fig. 12 dans laquelle nous découvrons une nouvelle fonctionnalité de **`gdal_warp`** : la fusion de plusieurs images en tenant compte éventuellement d'un masque de transparence. La commande pour obtenir cette image composite est limpide, avec la série des images à fusionner suivie de la sortie : **`gdalwarp tt_composite_export.tif warp_130723_0905.tif gdal_merge.tif`** avec **`warp_130723_0905.tif`** générée depuis QGis au moyen du module **`r.out.tiff`** appliqué à la couche issue de la correction géométrique par **`georeferencer`**. Le fichier de localisation, d'extension **`.tfw`**, est généré simultanément.

Bien que automatisé, le traitement d'images n'a de sens que sur des images nettes et convenablement illuminées. Nous avions déjà présenté une méthode classique de recherche d'images floues au moyen de l'auto-corrélation **[10]**. Pour rappel, l'auto-corrélation est une transformation mathématique visant à rechercher le décalage d'un motif maximisant le taux de ressemblance avec un motif de référence. Il semble évident qu'un motif se ressemble fortement à lui même et une auto-corrélation présente toujours un maximum pour le décalage (translation dans le cas d'une image) nul. Le point qui nous intéresse n'est pas tant la position de ce maximum trivial à l'origine, que le comportement de la courbe à son voisinage. Une image nette ne se ressemble à elle même et nous n'avons aucune raison de retrouver les caractéristiques spatiales des pixels – sauf en présence d'un motif périodique qui n'est en général pas le cas – lors d'une translation spatiale de l'image. Ainsi, une image nette présente un pic d'inter-corrélation pour un déplacement nul et une décroissance rapide au voisinage de ce maximum. Au contraire, une image floue se ressemble à elle même si elle est légèrement translatée. Le cas

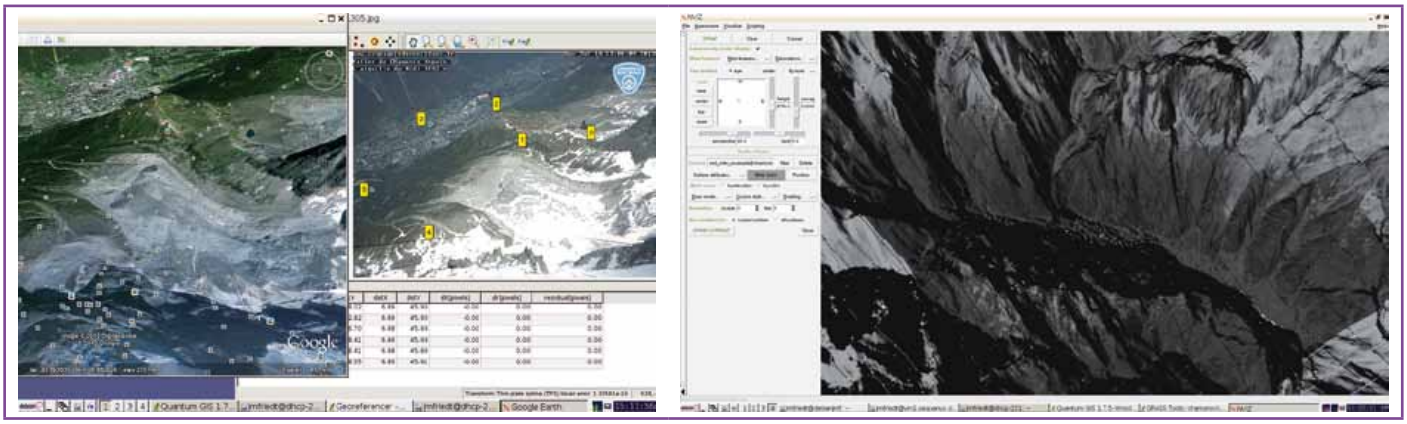


Figure 11 : Gauche : comparaison de la vue aérienne visible sur Google Earth (moitié gauche de la capture d'écran) et de la photographique prise par la webcam de la compagnie du Mont Blanc située au sommet de l'aiguille du Midi. Droite : drapage d'une image satellite de DigitalGlobe sur le MNE SRTM autour de la vallée de Chamonix, site de l'investigation en cours. De droite à gauche, les glaciers de Taconnaz, des Bossons et la mer de glace sont correctement positionnés sur la topographie. Le bout du glacier d'Argentière se devine à gauche de l'image.

classique que nous observons en pratique est un nuage qui englobe l'aiguille du Midi : dans ce cas l'image est uniformément grise et toute translation de l'image se ressemble à elle-même. Nous allons donc considérer la valeur de l'auto-corrélation bidimensionnelle en un point autre que la translation nulle et nous analyserons la valeur à une distance arbitrairement choisie de 10 pixels du centre. Cette procédure s'implémente sous GNU/Octave comme suit :

```
d=dir('13*1305.jpg'); % les images brutes de 13h05 en 2013
for k=1:length(d)
  d(k).name
  eval(['a=imread(''',d(k).name,''');']);
  aa=rgb2gray(a);
  aaa=aa(100:100+taille,100:100+taille);
  xx=xcorr2(aaa-mean(mean(aaa)),'coeff');
  s=size(xx);m=max(xx(:));
  p=find(xx(:)==m);
  [cx(k),cy(k)]=ind2sub(s,p);
  mm(k)=xx(cx(k)+10,cy(k));
end
kb=find(mm>0.5);   % les images ne sont pas bonnes
kg=find(mm<=0.5);  % les images sont bonnes
```

Nous nous proposons de justifier de l'acquisition des connaissances présentées jusqu'ici pour exploiter quantitativement ces photographies de webcams. La procédure de traitement que nous proposons est sur la Fig. 13. Il est probable qu'un utilisateur expérimenté saura effectuer toutes les opérations qui vont suivre avec les outils cités ci-dessus, mais étant plus familier avec GNU/Octave (ou Matlab), nous allons quitter le monde de la gestion d'informations spatiales pour revenir à des outils plus classiques de traitement du signal.

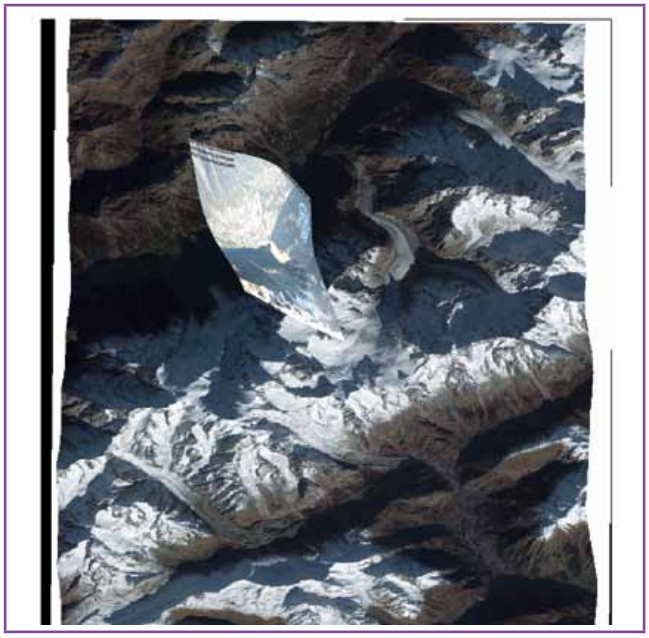


Figure 12 : gdal_warp permet non seulement de déformer une image mais aussi de fusionner plusieurs images afin soit d'assembler une mosaïque, soit comme ici, de fusionner deux ensembles de données couvrant des régions complémentaires. Noter la continuité des structures entre l'image de fond et l'image déformée, garantie de la validité de la déformation géométrique par un choix approprié des GCPs.

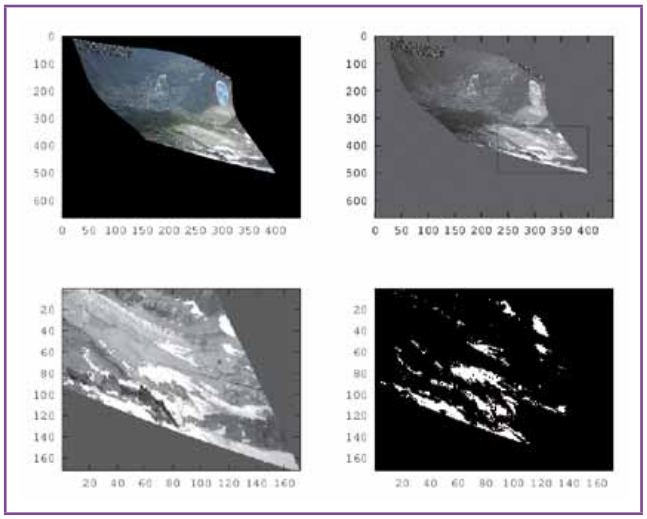


Figure 13 : De gauche à droite et de haut en bas : une image corrigée géométriquement brute en couleur ; passage en tons de gris et sélection d'un rectangle contenant le glacier des Pélerins ; les zones enneigées sont visibles comme des taches blanches en bas à gauche ; un seuillage permet de trouver les zones enneigées et de sommer les surfaces associées (en bas à droite).

Ayant obtenu une image déformée géométriquement pour tenir compte de la topographie du terrain sur laquelle est projetée l'image (Fig. 13, en haut à gauche), nous sélectionnons une région comprenant majoritairement le glacier que nous étudions (Fig. 13, en haut à droite). Bien qu'il soit envisageable sous GRASS de définir un masque complexe, nous nous contentons ici d'une région rectangulaire. Cette image est convertie en tons de gris puisque la couleur ne permet pas d'améliorer la capacité de différenciation entre roche, neige et glace. Les méthodes classiques d'analyses multispectrales, en particulier lorsqu'une bande infrarouge est disponible pour différentier les réflecteurs du rayonnement solaire (index NDVI – *Normalized Difference Vegetation Index* – combinaison normalisée de la réflectance dans les bandes infrarouge et visible) ne sont pas accessibles ici, et un rapide test d'analyse en composantes principales ne semble pas savoir faire mieux qu'une simple analyse en intensité lumineuse sur chaque pixel. En conséquent, un seuillage est effectué sur chaque pixel de l'image en tons de gris (Fig. 13, en bas à droite) en assignant une pondération de 1 pour un pixel brillant (au dessus d'un seuil) et de 0 en dessous. Finalement, la somme des pixels est effectuée et, puisque chaque pixel comporte une surface connue suite à la correction géométrique, nous sommes en mesure d'estimer la surface du glacier couverte de neige (et donc protégée de la fonte). L'implémentation de ces concepts sous GNU/Octave s'obtient comme suit :

```
seuil_neige=250
d=dir('./wa*1305.tif');  % les images corrigees
l=1;                     % geometriquement
for k=1:length(d)
  eval(['a=imread(''',d(k).name,''');']);
```

```
  d(k).name
  aa=rgb2gray (a);
  aaa=aa(330:500,230:400);
  seuil_neige=max(max(aaa));
  if (seuil_neige>120)   % en dessous de 120, ce n'est pas de la neige
      seuil_neige=seuil_neige-20; else  % on se laisse de la marge
      seuil_neige=seuil_neige+1;
  endif
  k=find(aaa>seuil_neige); % seuillage
  aaa(k)=0;k=find(aaa>0);aaa(k)=1;aaa=1-aaa; % 1 = neige, 0 sinon
%  if (l<=16)    % si on veut afficher les images seuillees
%     subplot(4,4,l);colormap gray
%     imagesc(aaa);colorbar;
%  endif
  somme(l)=sum(sum(aaa));
  l=l+1;
end
```

Le résultat de l'analyse d'images floutées ou non est présenté sur la Fig. 14, dans laquelle une série d'images identifiées comme floues est proposée à gauche et les images nettes à droite. L'analyse est uniquement effectuée sur la région de la photographie comportant le village de Chamonix (la cible présentant le plus de motifs et la plus lointaine de la caméra), le calcul d'inter-corrélation étant gourmand en ressources de calcul : une analyse sur une région restreinte suffit. Nous constatons que la classification est très efficace et ce, d'autant plus que nous avons choisi des horaires de prises de vue (13h05 et 15h05) maximisant le contraste des images. Un bref aperçu de Fig. 16, dont le graphique du haut donne la valeur normalisée de l'auto-corrélation à une distance de 10 pixels de la translation nulle, montre nettement la présence de deux classes, celles pour lesquelles l'auto-corrélation loin de l'origine est faible (image nette, valeur entre 0,4 et 0,5) et les images floues pour lesquelles cette valeur dépasse 0,5.

Ayant sélectionné les images dignes de traitement, nous découpons la région comportant le glacier (Fig. 15, gauche) et seuillons sur une valeur qui semble représentative de la présence de neige (Fig. 15, droite).

Finalement, les pixels blancs (valeur 1) et noirs (valeur 0) sont sommés pour estimer la couverture neigeuse : la conversion de pixel à surface s'obtient en analysant les méta-données d'une photographie corrigée géométriquement sous QGis. Nous y apprenons que chaque pixel occupe $1{,}81 \cdot 10^{-4}$ degrés/pixel, soit aux latitudes qui nous concernent, une surface de $14 \times 20 = 281$ m^2 :

```
$ gdalinfo warp_130724_0905.tif
Driver: GTiff/GeoTIFF
Files: warp_130724_0905.tif
Size is 446, 664
Coordinate System is ``
Origin = (6.841708607073000,45.956579737414486)
Pixel Size = (0.000181024830052,-0.000181024830052)
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (   6.8417086,  45.9565797)
Lower Left  (   6.8417086,  45.8363793)
Upper Right (   6.9224457,  45.9565797)
Lower Right (   6.9224457,  45.8363793)
Center      (   6.8820771,  45.8964795)
Band 1 Block=446x4 Type=Byte, ColorInterp=Red
  Mask Flags: PER_DATASET ALPHA
Band 2 Block=446x4 Type=Byte, ColorInterp=Green
  Mask Flags: PER_DATASET ALPHA
Band 3 Block=446x4 Type=Byte, ColorInterp=Blue
  Mask Flags: PER_DATASET ALPHA
Band 4 Block=446x4 Type=Byte, ColorInterp=Alpha
```

Sur la Fig. 15, la courbe du haut indique quelle photographie est digne d'étude (en bleu pour une séquence prise tous les jours à 13h05 entre les 14 juillet et 9 septembre, rouge pour la séquence prise à 15h05) – seule une valeur au-dessous de 0,5 indique une image nette – la courbe du milieu, l'analyse de la surface enneigée pour toutes les images indépendamment de leur qualité – fournissant une courbe difficile à analyser – et finalement l'analyse (en bas) uniquement pour les images dignes d'intérêt. Nous y observons une décroissance

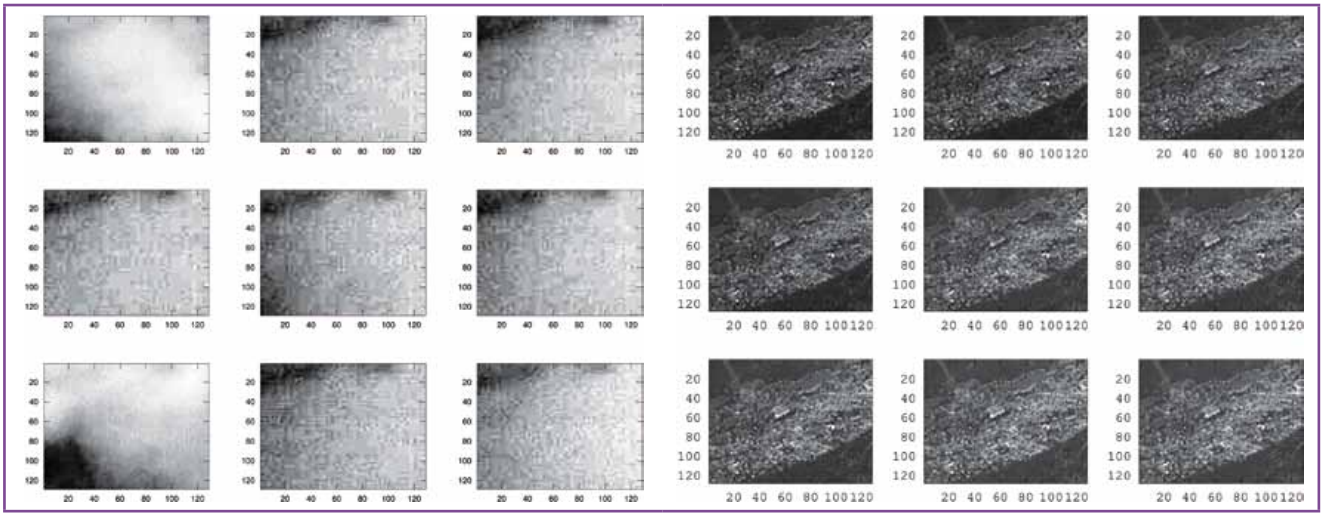


Figure 14 : Gauche : séquence d'images identifiées comme floues par une valeur d'auto-corrélation élevée à côté du point de décalage nul. Droite : séquence d'images identifiées comme nettes par inter-corrélation faible en dehors du décalage nul.

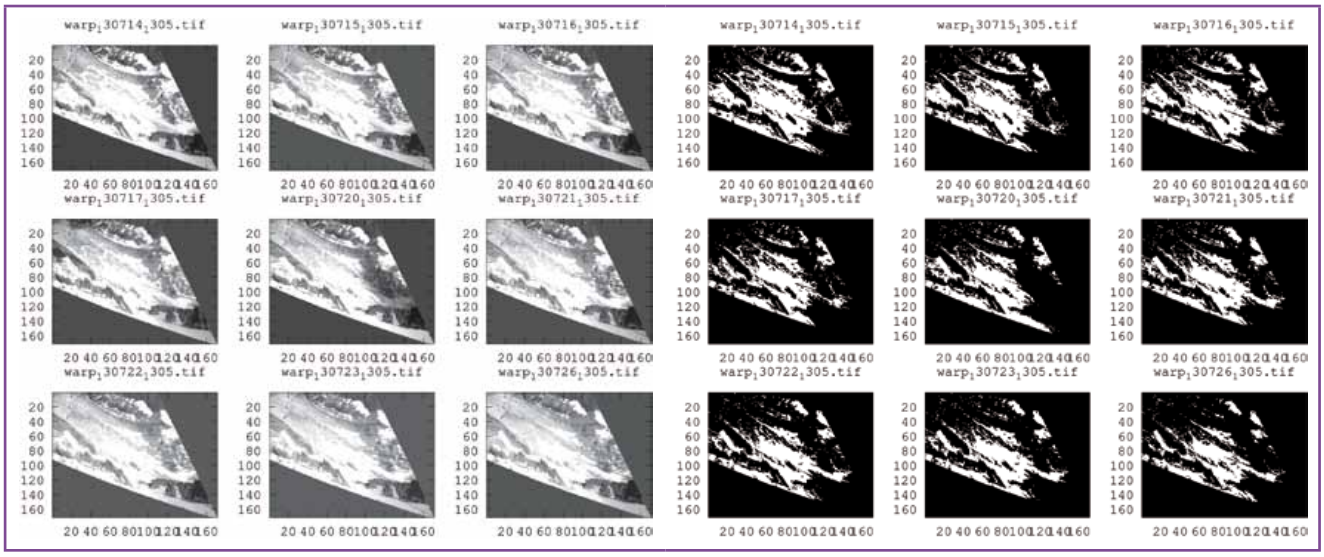


Figure 15 : Gauche : séquence d'images corrigées géométriquement et identifiées comme nettes, découpées autour du glacier des Pélerins. Droite : identification des zones couvertes de neige par seuillage.

monotone de la couverture neigeuse et une certaine cohérence entre les courbes obtenues avec les photographies acquises à 13h05 et 15h05. La valeur initiale de la courbe, 7000 pixels ou près de 2 km^2, dépasse la surface du glacier et indique clairement que des zones neigeuses en dehors du glacier sont comptabilisées. La valeur finale, 1000 pixels ou 0,3 km^2, est cohérente avec une fraction du glacier couverte de neige. Une validation sur le terrain serait évidemment nécessaire pour confirmer ces affirmations issues exclusivement de traitements d'images acquises par télémesure **[15]**.

Cette analyse, effectuée selon une approche naïve de seuil fixe par rapport au pixel d'intensité (somme des composantes rouge, bleue et verte) la plus forte pour définir la neige, s'avère la plus robuste face à des approches plus complexes de segmentation. Une classification par l'algorithme **`kmeans()`** (disponible dans la boite à outils **`statistics`** de GNU/Octave) donne certes un résultat esthétiquement agréable, mais l'assignation de la multitude de classes (neige à l'ombre, neige au soleil, glace à l'ombre, glace au soleil, roche) s'avère excessivement complexe.

```
d=dir(['./wa*_1105.png']);
for kk=1:length(d)      % application a toutes les images du repertoire
    eval(['a=imread(''./',d(kk).name,''');']);
    d(kk).name
    a=a(:,:,(1:3));     % elimine la couche de transparence
    aa=rgb2gray (a);    % couleur -> tons de gris
    aaa=aa(330:500,230:400); % se focalise uniquement sur le glacier
    aaaa=double(reshape(aaa,171*171,1)); % matrice -> vecteur
    k=find(aaaa>0);     % on elimine le fond noir de la correction ...
    b=aaaa(k);          %  ... geometrique par gdalwarp (sinon une classe de trop)
    [idx, centers] = kmeans (b, 3); % segmentation sur 3 classes
    aaaa(k)=idx;        % assignation des classes sur l'image originale
    bb=reshape(aaaa,171,171); % vecteur -> matrice
    imagesc(bb);        % affichage du resultat de la segmentation
end
```

À titre d'illustration, le traitement de la prise de vue de 13h05 du 8 octobre (Fig. 17) illustre clairement la discontinuité entre l'ombre projetée de l'aiguille du midi (en haut à gauche), le reflet sur la vitre de la webcam (en haut à droite) et la neige au premier plan (en bas), segmentation sans relation avec les motifs que nous recherchons.

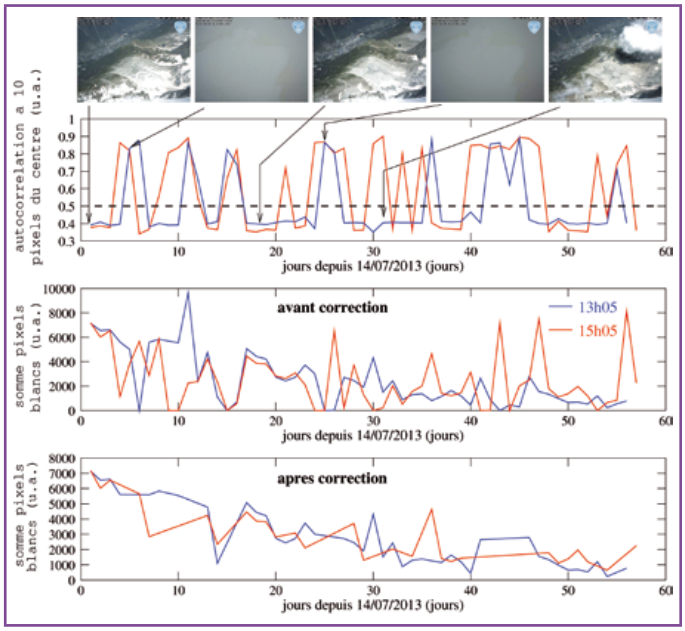


Figure 16 : De haut en bas : valeur de l'inter-corrélation en dehors du décalage nul (choisi arbitrairement à 10 pixels de l'origine) ; couverture neigeuse comme somme des pixels clairs dans l'image, sans sélection de la qualité de l'image traitée ; somme des pixels clairs comme représentative de la couverture neigeuse uniquement appliquée pour les photos prises par beau temps. L'analyse est dupliquée sur deux horaires de prises de vues quotidiennes – 13h05 et 15h05 – pour comparer la reproductibilité de la mesure et s'affranchir d'artefacts d'ombre projetée ou de couverture nuageuse.

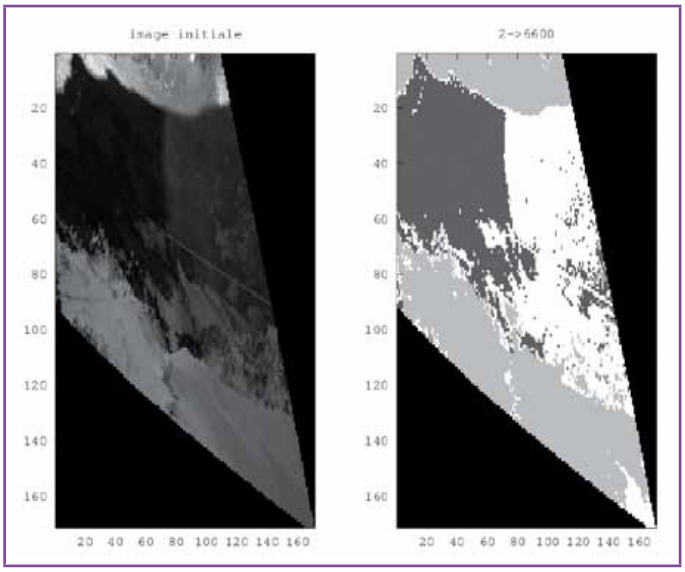


Figure 17 : Gauche : image brute obtenue le 8 octobre 2013 sur la webcam de la Compagnie du Mont Blanc située en haut de l'aiguille du Midi, convertie en tons de gris et déformée géométriquement. Droite : segmentation par algorithme de kmeans()configuré pour rechercher 3 classes. L'ombre projetée par la montage est clairement visible, mais ne correspond pas à l'objectif de notre étude. Seule une connaissance additionnelle de la position des ombres projetées permettra de s'affranchir de tels artefacts d'analyse.

7 Prévision des ombres projetées : une aide au traitement automatique des images

Un point critique pour classer de façon automatique les zones enneigées, englacées ou de roche tient en l'illumination. Il s'agit d'un problème bien connu en vision et traitement automatique des images : sans une bonne illumination homogène, le traitement numérique des images devient rapidement complexe. Dans notre cas, les ombres des nuages et autres artefacts météorologiques ne sont pas prévisibles mais nous pouvons au moins identifier les ombres projetées par le soleil, parfaitement déterministes. Connaissant les dates et horaires de prises de vues, nous connaissons la position du soleil et en déduisons, avec la topographie du site, le motif des ombres projetées (fonctions `r.sunmask` ou `r.sun` de GRASS). Ainsi, prédire la position des ombres et imposer la continuité des classes de part et d'autre des zones éclairées simplifie le problème d'identification de la couverture neigeuse des surfaces.

Nos essais sur un MNE en projection sphérique WGS84 se sont soldés par des échecs qui ont nécessité d'approfondir le problème jusqu'à trouver la solution que nous proposons ci-dessous, peut être pas optimale mais qui a le bon goût d'être fonctionnelle. Les cartes sont définies selon deux grandes classes de référentiels : non-projetées ou projetées. Dans le premier cas, un système de coordonnées sphérique définit la position de chaque point sur le globe – par exemple pour l'ellipsoïde utilisé par la majorité des récepteurs GPS – selon la convention WGS84 dans laquelle chaque point est défini par une longitude et une latitude en degrés. Cependant, ce mode de représentation des données (utilisé jusqu'ici) ne tient pas compte du fait qu'une carte ou un écran sont des objets plats, et qui par conséquent nécessitent une étape de projection d'une sphère vers un plan **[17]**. Le système de coordonnées projeté définit un certain nombre de régions – bandes selon les méridiens – pour lesquels la coordonnée sphérique est localement projetée sur un plan. Les diverses régions du globe ainsi définies tiennent compte des disparités locales entre l'ellipsoïde définissant WGS84 et la sphère idéale que pourrait représenter le monde si la Terre était homogène et n'était pas en rotation. Les conséquences évidentes de passer en référentiel projeté sont que :

- l'unité de distance n'est plus le degré mais le mètre. De cette façon, les distances sont plus faciles à estimer dans un référentiel projeté localement plan,
- la géométrie plane est plus facile à appréhender que la géométrie en coordonnées sphériques, ce qui aura une conséquence significative pour notre application.

Notre objectif est désormais de considérer la photographie acquise par la webcam de la Compagnie du Mont Blanc sur l'Aiguille du Midi le 22 Septembre 2013 à 13h05. Cette photographie présente une superbe ombre projetée qui interdit à peu près tout classement automatique de la couverture neigeuse sur le glacier. Si nous pouvons prédire la position de cette ombre projetée, nous pouvons ajouter un *a priori* additionnel sur l'algorithme de classification qu'est la continuité des classes de part et d'autre de l'ombre. Cependant, une application naïve de la fonction `r.sunmask` de GRASS qui prédit les ombres projetées par le soleil, sur un MNE en WGS84 échoue lamentablement. L'échec est illustré par l'indépendance du résultat du calcul avec l'élévation du soleil au-dessus de l'horizon, résultat évidemment aberrant puisqu'un soleil rasant doit induire de longues ombres projetées alors qu'un soleil au zénith n'induit aucune ombre. Diverses expérimentations nous ont permis de conclure qu'il est nécessaire de passer en coordonnées projetées (sur un plan local) au lieu de travailler en coordonnées sphériques. Pour l'Est de la France, la zone de projection se nomme WGS84/UTM31N. Nous devrons donc convertir l'ensemble des données en notre possession dans ce référentiel pour y appliquer les algorithmes de prévision des ombres projetées. Afin de rendre une longue histoire courte, notamment avec une description des expériences sur données simulées qui nous a amené à cette conclusion, le lecteur est renvoyé vers la page web http://jmfriedt.free.fr/proj_grass/projection.html.

Figure 18 : De gauche à droite : image obtenue le 22 Septembre 2013 avec les ombres projetées de deux montagnes ; prévision par la fonction r.sunmask de GRASS des ombres projetées sur un MNE issue de SRTM donc avec une résolution apparemment insuffisante pour résoudre les deux pics (élévation du soleil de 44,2°, azimuth 174,7°) ; superposition des deux résultats illustrant une concordance des résultats excellente compte tenu de la précision avec laquelle la photographie est localisée spatialement.

Le résultat du calcul des ombres projetées, une fois toutes les données converties au format projeté WGS84/UTM31N, est présenté Fig. 18. Pour atteindre ce résultat, nous avons exploité la version 2.0.1 de QGis qui permet de passer d'un mode de projection à un autre simplement par l'application de la commande ***Save As*** .... Ainsi, nous avons défini une carte en référentiel projeté WGS84/UTM31 pour QGis, y avons chargé le MNE SRTM en indiquant que le format d'entrée est WGS84 (non-projeté) puis en sauvant ces données au nouveau format. Travaillant de même dans GRASS, une carte au format projeté WGS84/UTM31N est définie et le fichier que nous venons de sauvegarder y est inséré (**`r.in.gdal`**) en appliquant l'option d'ignorer le mode de projection (*override projection*). Ainsi, GRASS charge le MNE dans la projection appropriée et permet d'appliquer l'algorithme de traitement **`r.sunmask`** dans un référentiel approprié pour atteindre un résultat dépendant de l'élévation du soleil au-dessus de l'horizon. Finalement, la carte résultante, qui contient des valeurs arbitraires (NaN) pour les ombres projetées et des valeurs scalaires finies, est affichée en sélectionnant une transparence de l'ordre de 65% pour l'ensemble de la carte (mode *Singleband pseudocolor*) et une valeur de 100% pour les valeurs non-définies (penser à activer la bande spectrale 1 (Gray) comme sélection de la transparence, Fig. 19). La position du soleil, connaissant la position géographique de l'observateur et la date de prise de vue, est obtenue depuis le site de l'USNO[8]. Nous prendrons un soin particulier à réduire au strict minimum la surface analysée – au moyen de la définition de la zone d'analyse de GRASS auquel le MNE est ajusté par **`r.resample`** – car l'application de **`r.sunmask`** sur une large région avec une résolution égale à celle du MNE est excessivement gourmande en ressources de calcul.

L'application de l'algorithme de **`r.sunmask`**, gourmande en ressources calculatoires, ne semble pas savoir profiter de la disponibilité de plusieurs processeurs pour paralléliser ses opérations. Un calcul sur 1700×2300 points, pour une résolution spatiale de 10×10 m^2, prend un peu moins de 72 heures. Il semble donc pertinent de s'interroger sur la résolution requise pour obtenir un résultat exploitable quant à l'ombre projetée. Le sommet de l'aiguille du Midi se trouve à une altitude de 3842 m tandis que le glacier des Pélerins présente un front à une altitude de l'ordre de 2400 m. La différence d'altitude entre la source de l'ombre projetée (supposée être l'aiguille

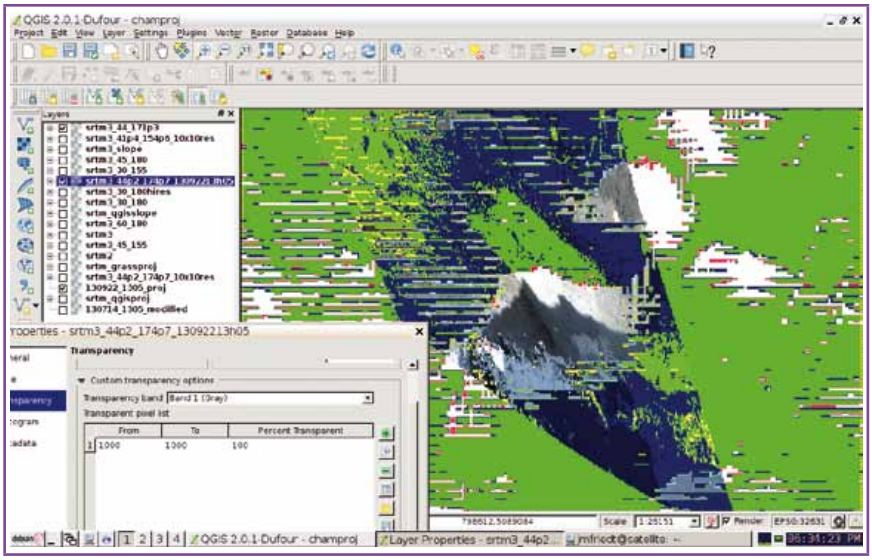


Figure 19 : Zoom sur la zone ombrée prévue par la topographie et la position du soleil et configuration de la transparence pour rendre les zones ombrées visibles. Dans le menu Style, nous activons simplement Singleband Pseudocolor pour sélectionner la couleur des zones éclairées en cliquant sur "+" et sélectionnant une unique classe de valeur 0.

[8] http://aa.usno.navy.mil/data/docs/AltAz.php

du Midi) et la surface du glacier est donc 1442 m. Puisque l'angle du soleil autour du mois de Septembre est environ de 45°, l'ombre projetée est elle même longue de l'ordre de 1442 m. Détecter une variation de 1° de la position du soleil nécessite une résolution de 50 m puisque l'ombre projetée pour des angles de 44 et 46° est 1392 et 1493 m respectivement. Le calcul de la position du soleil au moyen du site de l'USNO pour quelques jours autour du 22 Septembre 2013, pour des prises de vues à heure fixe autour de 13h (heure local), indiquent que l'élévation du soleil varie de l'ordre de 0,4°/jour. Ainsi, une résolution spatiale de 30 m ne permettra que d'identifier à deux jours près la date de prise de vue, en supposant que le pixel de calcul soit significatif. Une identification de la date de prise d'une photo par la méthode de l'ombre projetée à la semaine près semble plus réaliste compte tenu de la résolution spatiale du MNE exploité ici (90 m annoncés à l'équateur pour SRTM).

Conclusion

S'étant familiarisé avec les outils de traitement spatial de l'information au moyen de QGis en faisant appel aux fonctions des bibliothèques GRASS et GDAL, nous nous sommes efforcés de nous affranchir de l'interface graphique en identifiant les lignes de commandes associées à chaque opération, obtenant ainsi une série de traitements parallélisables depuis un script sur un grand nombre d'images.

Nous nous sommes efforcés de démontrer l'utilité de ce concept pour le traitement quantitatif d'informations obtenus sur des images acquises sur des prises de vues obliques en exploitant les photographies acquises par une webcam dans le massif du Mont Blanc. La série de photographies ainsi corrigées géométriquement est utilisée pour estimer la couverture neigeuse sur un petit glacier alpin, après élimination des clichés inexploitables du fait de conditions météorologiques défavorables.

Au cours de ces analyses de séquences d'images, nous avons fait l'hypothèse que les points de contrôle identifiés sur une image de référence restent valables pour toute la séquence de prises de vues. Cette hypothèse devient erronée si la caméra est sujette à des petits déplacements, qu'il s'agisse de maintenance ou de l'effet du vent sur l'attache de la caméra sur son support qui la fait vibrer. L'observation de structures caractéristiques sur l'image doit permettre d'identifier ces mouvements et donc de les corriger, les conditions de prises de vues (grossissement de l'objectif, aberrations optiques) restant constantes. Ainsi, le point que nous n'avons pas abordé concerne l'identification préalable de la rotation et translation de photographies par rapport à une image de référence (*registration*) – peut être au moyen du module i.homography de GRASS **[16]**. Le classement à posteriori des diverses surfaces visibles sur la mosaïque des images (classification neige/glace) n'est ici que grossier avec un seuillage et une classification plus fine mériterait à être analysée en détail au moyen des outils dédiés proposés par GRASS (classification supervisée ou non[9]). ■

Remerciements

Les équipes de géographes des projets financés par l'Agence National de la Recherche (ANR) Hydro-Sensor-Flow et Cryo-Sensors m'ont introduit au problème du traitement de données géographiques et de l'exploitation des images en prises de vues obliques. A. Sauter (Univ. Paris I) m'a montré l'inadéquation du MNE GDEM2 et la nécessité d'utiliser SRTM pour un rendu visuel acceptable sous nviz. J.-P. Simonnet (labo. Chronoenvironnement, Besançon) m'a fourni une série d'images prises par ULM, dont une est utilisée sur la Fig. 5. Y. LeNir (EISTI, Pau) a mentionné l'efficacité de l'algorithme `kmeans()` pour segmenter une image. Le problème de la prévision des ombres projetées pour faciliter la classification des surfaces a aussi été formalisé au cours de cette discussion. Les références bibliographiques qui ne sont pas identifiables par une recherche sur le web ont été obtenues auprès de Library Genesis (`gen.lib.rus.ec`).

Références

[1] J.G. Corripio, Y. Durand, G. Guyomarc'h, L. Mérindol, D. Lecorps & P. Pugliése, Land-based remote sensing of snow for the validation of a snow transport model, Cold Regions Science and Technology 39 (2004), pp.94–104 (http://www.uibk.ac.at/geographie/personal/corripio/publications/corripioetal04_crst.pdf) ou http://www.meteoexploration.com/products/monitoring.html

[2] V. Lebourgeois, A. Bégué, S. Labbé, B. Mallavan, L. Prévot & B. Roux, Can Commercial Digital Cameras Be Used as Multispectral Sensors ? A Crop Monitoring Test, Sensors (8), 7300–7322 (2008), disponilbe à www.mdpi.com/1424-8220/8/11/7300

[3] i.ortho.photo est décrit à http://grasswiki.osgeo.org/wiki/Orthorectification_digital_camera

[4] E. Thibert, C. Vincent, A. Soruco, M. Harter, R. Blanc, R. Heno, Photogrammétrie numérique & risques naturels : Application à la dynamique des avalanches et aux chutes de séracs – Synthèse effectuée pour le Pôle Grenoblois Risques Naturelsdisponible à http://www.risknat.org/pages/programme_dep/docs/cemagref_etna/2009_Thibert-et-Vincent-jan11.pdf, ou C. Vincent, E. Thibert & M Harter, Etude du glacier de Taconnaz, disponible àhttp://www.glariskalp.eu/files/Action_2.A%20-%20Etude_du_glacier_de_Taconnaz.pdf

[9] http://www.portailsig.org/content/classification-d-images-superviseenon-supervisee-sous-grass

[5] J.G. Corripio, Snow albedo estimation using terrestrial photography, Int. J. Remote. Sensing, 25 (4), pp.5705-5729 (2004)

[6] D. Laffly, É. Bernard, M. Griselin, F. Tolle, J-M. Friedt, G. Martin & C. Marlin, High temporal resolution monitoring of snow cover using oblique view ground-based pictures, Polar Record 48 (1), pp 11-16 (Jan. 2012), disponible à http://jmfriedt.free.fr/S0032247411000519a.pdf

[7] J.-M Friedt, É. Carry, Acquisition et dissémination de trames GPS à des fins de cartographie libre, GNU/Linux Magazine France, Hors Série 27 (Octobre 2006)

[8] W.G. Rees, Assesment of ASTER Global Digital Elevation Model data for Arctic Research, Polar Record 48 (1) pp. 31-39 (Jan. 2012)

[9] V. Kaufmann, Measurement of surface flow velocity of active rock glaciers using orthophotos of virtual globes, Geographia Technica, Special Issue, pp. 68-81 (2010)

[10] J.-M Friedt, Auto et intercorrélation, recherche de ressemblance dans les signaux : application à l'identification d'images floutées, GNU/Linux Magazine France 139 (Juin 2011), disponible à http://jmfriedt.free.fr/xcorr.pdf

[11] P.T. Fretwell, M.A. LaRue, P. Morin, G.L. Kooyman, B. Wienecke, N. Ratcliffe, A.J. Fox, A.H. Fleming, C. Porter & P.N. Trathan, An emperor penguin population estimate: the first global, synoptic survey of a species from space, PloS one 7 (4), e33751 (2012), disponible à http://www.plosone.org/article/info%3Adoi%2F10.1371%2Fjournal.pone.0033751

[12] H.B. de Saussure, Voyages dans les Alpes: partie pittoresque des ouvrages (3ème édition), Ed J. Cherbuliez, (1855), disponible à http://doc.rero.ch/record/17091/files/CA_42.pdf

[13] D. Farinotti, J. Magnusson, M. Huss & A. Bauder, Snow accumulation distribution inferred from time-lapse photography and simple modelling, J. Geophys. Research (2010), p.F01014

[14] R. Vivian, Les glaciers du Mont-Blanc, Ed. La Fontaine de Siloé (2005) [10]

[15] R. Fallourd, Suivi des glaciers alpins par combinaison d'informations hétérogènes : images SAR Haute Résolution et mesures terrain, thèse de l'Université de Grenoble (2012), disponible à http://tel.archives-ouvertes.fr/tel-00718596/

[16] i.homography est décrit à http://www.cp-idea.org/documentos/tecnologia/7_grassgisconference.pdf et disponible à http://grasswiki.osgeo.org/wiki/GRASS_AddOns#Imagery_add-ons.

[17] J. Lefort, L'aventure cartographique, Belin – Pour la Science (2005) ou J.-P. Snyder, Flattening the Earth - Two Thousand Years of Map Projections, The University of Chicago Press (1993)

[10] http://www.geologie-montblanc.fr/glaciers.html

LOG4PHP : ADOPTEZ UNE MÉTHODOLOGIE RIGOUREUSE POUR LES LOGS DE VOTRE APPLICATION WEB

par Stéphane Mourey [Taohacker]

Tout d'abord, il y eut log4j, un framework Java dédié à la génération de logs depuis une application. Ce projet, repris par la Fondation Apache a depuis essaimé vers d'autres langages dont PHP avec log4php, projet de la même fondation. De quoi mettre un peu de rigueur dans vos logs.

La production d'informations sur le fonctionnement d'une application est importante à plus d'un titre : au développeur tout d'abord, pour vérifier le fonctionnement interne de son script et y découvrir des erreurs ; à l'administrateur ensuite, pendant la vie de l'application pour en vérifier la bonne santé ; parfois aux deux à la fois, lorsqu'il s'agit de comprendre un fonctionnement anormal de l'application durant son exploitation. Ces trois situations, bien que répondant à des contraintes différentes, ont ceci de commun qu'elles se caractérisent par l'introduction dans le code, ici ou là, d'une ligne dont la seule fonction est de signaler un événement et d'extraire des informations sur l'état interne de l'application. Puis ces informations sont envoyées à des destinataires différents selon la situation : affichage dans la page web, écriture dans un fichier, mail... Mais comme les logs ne sont pas destinés au client, votre commercial vous fera remarquer qu'il ne paie pas pour cela et le temps qu'il sera possible de leur consacrer sera minime. Trop souvent, cela se résumera à une ligne à décommenter pour provoquer l'affichage d'une variable dans la page.

Tout cela peut changer avec log4php. Le but de log4php est de permettre une gestion rigoureuse de ces flux d'événements quelle que soit la situation, en conservant un code toujours aussi simple pour déclencher l'enregistrement de l'événement. Le traitement de l'affichage ou de l'enregistrement est déporté dans la configuration. Le développeur met en place une configuration qui convient à son débogage, l'administrateur intégrera ces logs dans ses propres outils de surveillance, et une configuration adaptée à la situation pourra être mise en place au besoin pour analyser un fonctionnement anormal sans toucher à l'application.

1 Installation

La méthode à privilégier pour l'installation est sans conteste de passer par Composer. Si vous n'êtes pas familiarisé avec cette méthode, vous pourrez vous reporter utilement à notre article sur le sujet « Enfin une gestion de dépendances correcte pour PHP : Composer » paru dans GNU/Linux Magazine N° 162 du mois de juillet dernier.

Si votre projet ne comporte pas de fichier **composer.json**, vous devrez en créer un à la racine de votre projet ; sinon, ajoutez ces quelques lignes en adaptant l'imbrication des accolades à votre contexte :

```
{
    "require": {
        "apache/log4php": "2.3.0"
    }
}
```

Puis, ouvrez un shell au même emplacement, et lancez la commande suivante :

```
$ php composer.phar install
```

Si vous n'en avez pas encore, le dossier **vendor** sera créé, contenant divers éléments liés à Composer et puis un dossier apache, contenant lui-même un dossier **log4php** : c'est là que se trouve notre bibliothèque.

Tout cela suppose naturellement que Composer soit installé. Si vous ne voulez vraiment pas en passer par là, vous pourrez suivre des instructions plus habituelles sur la page du site officiel dédié à l'installation (https://logging.apache.org/log4php/install.html), soit par téléchargement et décompression, soit en utilisant PEAR.

2 Première utilisation

2.1 Charger la bibliothèque

Pour pouvoir utiliser cette librairie, il faut naturellement la charger. Si vous utilisez Composer, ce sera chose facile, si cela n'est pas déjà fait, puisqu'il suffit d'en charger l'autoloader en ajoutant simplement la ligne :

```
require 'vendor/autoload.php';
```

Sinon, il faudra inclure le fichier **Logger.php** qui se trouve dans **src/main/php** à partir de l'emplacement où vous avez décompressé le paquet :

```
require 'log4php/src/main/php/Logger.php';
```

2.2 Premiers événements

Partant de là, rien ne vous empêche plus d'écrire vos premières lignes de logs, la configuration par défaut répondant par un affichage sur **stdout**, c'est à dire votre console. Essayons avec un premier fichier **test.php** :

```
<?php
require 'vendor/autoload.php';
$logger = Logger::getLogger("main");
$logger->trace('Juste une petite note en passant...');
$logger->debug('Pour les yeux du développeur !');
$logger->info('Jusqu\'ici, tout va bien...') ;
$logger->warn('Quelque chose d\'anormal vient de se produire !');
$logger->error('Je commence à m\'emmêler les pinceaux....');
$logger->fatal('Je me meurs, je suis mort, je suis enterré !');
```

Nous commençons par récupérer un objet **logger**. Le **logger** est l'objet nommé qui prend en charge votre requête de nouvelle entrée de journal (ou requête de log) et qui la traite. Nous l'appelons à l'aide de la méthode **Logger::getLogger**, qui prend comme argument le nom du **logger**. Si celui-ci n'existe pas encore, il est créé. Il est ainsi possible d'utiliser plusieurs loggers dans votre code, chacun avec sa propre configuration...

Puis, nous lançons différents événements. La méthode utilisée pour lancer un événement dépend de sa gravité, en allant, comme dans notre exemple, de la simple trace sans beaucoup d'importance, simple information pour le développeur à l'erreur fatale, supposée être accompagnée d'un interruption de l'exécution en cours.

3 Configuration

Maintenant que nous avons découvert les méthodes permettant d'enregistrer des événements dans un logger, encore faut-il pouvoir les configurer pour que ce logger réagisse de la façon que nous voulons. Pour ce faire, log4php utilise des fichiers de configuration XML ou des configurations PHP qui lui donnent tout sa puissance. Mais auparavant, il nous faut définir...

3.1 Quelques concepts

Il n'y en a que deux nouveaux. Nous avons déjà vu le logger, l'objet qui prend en charge l'événement levé. Il est susceptible d'utiliser deux autres composants : les appenders et les layouts.

Commençons par les plus simples : les layouts. Il s'agit tout simplement de composants définissant le format à utiliser pour enregistrer l'événement. Ils permettent une conversion de l'événement en chaîne de caractères, pouvant être un texte simple, du HTML, du XML... Vous en trouverez le détail à cette url : http://logging.apache.org/log4php/docs/layouts.html.

Les appenders définissent la destination de l'enregistrement qu'il s'agisse de la console, d'un fichier, d'une base de données, d'un socket réseau... Un logger est susceptible d'utiliser plusieurs appenders et pour chacun d'eux le layout approprié.

3.2 Fichiers XML ou tableaux PHP ?

À votre guise, selon votre besoin. L'avantage d'utiliser des fichiers XML est de séparer cette configuration à l'extérieur du code PHP, ce qui est sans doute plus propre. Par contre, si vous n'avez pas la maîtrise de la configuration de votre serveur, ou si vous souhaitez configurer vos loggers de manière dynamique, l'utilisation de tableaux PHP est sans doute plus appropriée. SI vous optez pour des fichiers XML, assurez-vous que vos visiteurs ne peuvent y avoir accès (via un **.htaccess** sous Apache, par exemple), car vos loggers sont susceptibles de contenir des informations importantes sur votre infrastructure réseau, et en particulier, un attaquant qui y aurait accès connaîtrait en détail au moins un moyen par lequel vous prenez connaissance de l'état interne de votre application, ce qui n'est pas rien pour mener des actions en toute discrétion.

L'intérêt de la configuration est de définir les liens qui unissent les trois éléments : logger, appender et layout.

Un exemple de configuration XML :

```
<?xml version="1.0" encoding="UTF-8"?>
  <configuration xmlns="http://logging.apache.org/log4php/">
  <appender name="default" class="LoggerAppenderFile">
    <layout class="LoggerLayoutSimple" />
    <param name="file" value="/var/log/my.log" />
    <param name="append" value="true" />
  </appender>
  <root>
  <appender_ref ref="default" />
  </root>
</configuration>
```

Ici, nous configurons un appender vers le fichier **/var/log/my.log**. que nous affectons ensuite au logger racine.

Pour charger cette configuration dans un objet logger PHP, on utilisera la méthode **Logger::configure** :

```
Logger::configure('config.xml');
```

Voici maintenant la même configuration, mais en utilisant un tableau PHP cette fois-ci :

```
Logger::configure(array(
  'rootLogger' => array(
    'appenders' => array('default'),
  ),
  'appenders' => array(
    'default' => array(
      'class' => 'LoggerAppenderFile',
      'layout' => array(
        'class' => 'LoggerLayoutSimple'
      ),
      'params' => array(
        'file' => '/var/log/my.log',
        'append' => true
      )
    )
  )
));
```

Vous remarquerez le parallèle qui existe entre les deux structures, ce qui vous permettra de passer aisément de l'un à l'autre. Mais pour éviter tout risque d'erreur, une méthode existe pour traduire un fichier XML en un tableau PHP :

```
$configurator = new LoggerConfiguratorDefault();
$config = $configurator->parse('/path/to/config.xml');
```

Par contre, il n'y a pas de méthode automatique pour procéder à l'inverse.

3.3 Loggers

Nous ne cessons d'évoquer les loggers, il nous faut approfondir leur concept de façon à pouvoir en tirer pleinement partie. Nous avons dit que plusieurs loggers pouvaient exister au sein de votre application, chacun étant accessible par son nom. Ajoutons maintenant qu'une pratique courante est de donner à chaque classe d'objet son propre logger en utilisant le nom de la classe comme nom pour le logger. Cela permet de focaliser les logs sur tel ou tel type d'objet.

Il est possible d'affecter un seuil à un logger. Celui-ci ne réagira alors que si le requête de log dépasse ou égale ce seuil. Par exemple, pour ignorer les requêtes inférieures au niveau **INFO**, soit les niveaux **TRACE** et **DEBUG**, on utilisera la directive **level**. On obtient alors une configuration semblable à celle-ci :

```
<configuration xmlns="http://logging.apache.org/log4php/">
  <appender name="default" class="LoggerAppenderConsole" />
  <root>
    <level value="info" />
    <appender_ref ref="default" />
  </root>
</configuration>
```

Les loggers sont susceptibles d'être structurés hiérarchiquement selon une convention de nommage. On parlera alors de parents et d'enfants, car des mécanismes d'héritage sont à l'œuvre (à ne pas confondre avec l'héritage de classes, nous parlons ici d'héritage entre des objets instanciés). La convention pour définir un logger comme étant l'enfant d'un autre est de prendre le nom de son parent, d'ajouter un point, puis d'ajouter encore le nom de l'enfant. On obtient alors un nom de la forme : **parent.enfant**. Dès lors, l'enfant hérite de la configuration concernant le seuil et les appenders depuis son parent, sauf dans la mesure où vous la modifiez exceptionnellement.

Il existe un logger racine au sommet de la hiérarchie appelé **root**. Il a ceci de particulier qu'il existe toujours et ne peut être accédé par son nom, il faut utiliser la méthode **Logger::getRootLogger** à la place. Il est possible ainsi de modifier son configuration pour que tous les autres loggers en héritent.

3.4 Appenders

Vous aurez peut-être remarqué dans l'exemple précédent que les appenders doivent être configurés à part dans le fichier XML, et que ce n'est qu'ensuite qu'ils sont appelés par leur nom au sein de la configuration des loggers. Cela permet de ne les configurer qu'une seule fois pour pouvoir les utiliser aussi souvent que nécessaire.

Prenons quelques instants pour lister les différentes classes de logger les plus importantes, car ce mécanisme de distribution est pour beaucoup dans l'intérêt de cette librairie :

- **LoggerAppenderConsole** : affiche l'événement sur la console,
- **LoggerAppenderEcho** : utilise la fonction PHP **echo**,
- **LoggerAppenderFile** : envoie l'événement dans un fichier,
- **LoggerAppenderDailyFile** : idem, mais avec un nouveau fichier pour chaque jour,
- **LoggerAppenderRollingFile** : idem, mais un nouveau fichier ne sera utilisé que lorsque le précédent aura atteint une certaine taille,
- **LoggerAppenderMail** : envoie tout le journal dans un mail,
- **LoggerAppenderMailEvent** : envoie un mail pour chaque événement,
- **LoggerAppenderPDO** : utilise une base de données PDO,
- **LoggerAppenderPhp** : crée un message d'erreur PHP de niveau utilisateur en utilisant la fonciton **trigger_error**.
- **LoggerAppenderSyslog** : ajoute une entrée au journal système.

Voilà déjà de quoi s'amuser !

La configuration d'un appender comprend plusieurs éléments :

- un nom unique pour l'identifier par la suite : **name**,
- une classe, parmi celles listées ci-dessus, qui permet de définir son fonctionnement : **class**,

- un layout, qui définit la format à utiliser pour enregistrer les événements,
- puis, éventuellement, d'autres paramètres, dépendants de la classe de l'appender et permettant de définir finement son comportement.

Le nom et la classe sont des attributs de la balise **appender**. Le layout est défini par l'intermédiaire d'une balise incluse **layout**. Les paramètres sont définis par des balises incluses **param** ayant deux attributs : **name** pour le nom du paramètre et **value** pour sa valeur. Voici un nouvel exemple comportant un appender un peu plus élaboré :

```
<configuration xmlns="http://logging.apache.org/log4php/">
  <appender name="default" class="LoggerAppenderFile">
    <layout class="LoggerLayoutSimple" />
    <param name="file" value="/var/log/my.log" />
    <param name="append" value="true" />
  </appender>
  <root>
    <appender_ref ref="default" />
  </root>
</configuration>
```

Ici, nous avons un appender ayant pour nom **default** de classe **LoggerAppenderFile** qui va ajouter les événements dans le fichier **/var/log/my.log**.

Remarquez la façon dont le lien est tissé entre le logger et l'appender : on ajoute une balise **appender_ref**, incluse dans celle du logger, avec un attribut **ref** qui prend comme valeur le nom de l'appender à associer.

Enfin, ajoutons que, comme pour le logger, il est possible de définir un seuil en-dessous duquel l'appender restera inopérant. Cela renforce l'intérêt d'utiliser plusieurs appenders pour un seul logger : ce dernier est alors en mesure de distribuer des événements en fonction de leur gravité.

Par contre, petite inconsistance dans le format de configuration, l'appender n'utilise pas la balise **level** comme le fait le logger. La définition du seuil d'alerte s'effectue à l'aide de l'attribut **theshold** de la balise **appender**. Par exemple :

```
<appender name="default" class="LoggerAppenderEcho" threshold="WARN" />
```

3.5 Layouts

Les layouts permettent donc de définir les formats à utiliser pour enregistrer les événements dans un appender. En voici les classes les plus intéressantes :

- **LoggerLayoutHTML** : utilise un tableau HTML
- **LoggerLayoutPattern** : utilise une chaîne de caractères comme motif, ce qui lui donne une grande souplesse,
- **LoggerLayoutXml** : utilise un format XML.

Chaque layout possède ses propres paramètres de configuration, bien que certains puissent être communs à plusieurs classes comme **locationinfo** qui permet d'indiquer à certains layouts que les informations quant à l'endroit du code (fichier, ligne, classe, méthode) où l'événement s'est produit doivent être intégrées au journal.

Voici un exemple de configuration d'appender utilisant la commande **echo** avec un événement formaté selon un motif :

```
<appender name="default" class="LoggerAppenderEcho">
  <layout class="LoggerLayoutPattern">
    <param name="conversionPattern" value="%date{Y-m-d H:i:s} %logger
%-5level %msg%n" />
  </layout>
</appender>
```

Ici, le motif "**%date{Y-m-d H:i:s} %logger %-5level %msg%n**" comprend tout d'abord la date suivant le schéma entre accolades, le nom du logger, le niveau de l'alerte sur au moins cinq caractères (pour conserver un alignement cohérent), puis le contenu du message et enfin un retour à la ligne.

Vous trouverez la référence complète de la description du motif à l'adresse suivante : http://logging.apache.org/log4php/docs/layouts/pattern.html.

4 Un pas plus loin

4.1 Filtres

Vous voilà maintenant armé pour logguer tout ce que vous souhaitez. Mais il se pourrait que vous rencontriez le problème inverse : vous pouvez vous retrouver avec trop d'événements loggués et des difficultés pour faire le tri. Pour cela, il y a les filtres.

Les filtres peuvent être associés à n'importe quel appender. Plusieurs peuvent même être définis, et ils forment alors une chaîne. Chaque filtre doit prendre une décision en examinant l'événement à logguer. Trois possibilités existent pour cette décision : accepter l'événement, auquel cas il est loggué sans que le reste de la chaîne des filtres soit examiné ; refuser l'événement, là, l'événement n'est pas loggué et ce sans autre considération ; enfin, rester neutre, alors l'événement est passé au filtre suivant de la chaîne.

Il existe quatre types de filtres, chacun ayant ses propres options de configuration :

- **LoggerFilterDenyAll** : refuse tous les événements,
- **LoggerFilterLevelMatch** : refuse ou accepte tous les événements d'un niveau précis et pas des autres niveaux, qu'ils soient inférieurs ou supérieurs,
- **LoggerFilterLevelRange** : semblable au précédent, n'en diffère qu'en proposant un intervalle de niveaux pour les événements filtrés, pas seulement un niveau précis,
- **LoggerFilterStringMatch** : accepte ou refuse les événements dont le message contient une chaîne de caractères.

Pour ma part, je regrette l'absence d'un filtre s'appuyant sur une expression régulière...

Une option commune à tous les filtres est **acceptOnMatch** qui permet de définir le sens du filtre. Si cette option est positionnée à **true** (ce qui est le cas par défaut), alors seul les événements qui correspondent au filtre sont acceptés ; à l'inverse, s'il est positionné à **false**.

Voici un exemple de configuration utilisant largement les filtres :

```
<configuration xmlns="http://logging.apache.org/log4php/">
  <appender name="default" class="LoggerAppenderEcho">
    <layout class="LoggerLayoutSimple"/>
    <filter class="LoggerFilterStringMatch">
      <param name="stringToMatch" value="intéressant" />
      <param name="acceptOnMatch" value="true" />
    </filter>
    <filter class="LoggerFilterLevelRange">
      <param name="levelMin" value="debug" />
      <param name="levelMax" value="error" />
    </filter>
  </appender>
  <root>
    <level value="TRACE" />
    <appender_ref ref="default" />
  </root>
</configuration>
```

4.2 Renderers

L'un des intérêts de log4php est qu'il permet également de logguer les propriétés d'un objet. Il peut le faire de manière extrêmement simple, sans intervention complexe du programmeur. Voyez le code suivant :

```
<?php
class magazine {
  public $number ;
  public $title ;
}
$linuxMag = new magazine();
$linuxMag->number = 'HS 67;
$linuxMag->title = 'MySQL & Bases de données';
$logger = Logger::getLogger('main');
$logger->info('Attention, nous allons logguer un objet');
$logger->info($linuxMag);
```

Il suffit de passer l'objet à la méthode de log ! Pour rendre l'objet, log4php utilise un renderer. Le code ci-dessus, sauf configuration spéciale, utilisera le renderer par défaut, et du coup une sortie qui ressemblera à ceci :

```
INFO - Attention nous allons logguer un objet
INFO - Magazine Object
(
  [number] => HS 67
  [title] => MySQL & Bases de données
)
```

log4php permet de créer son propre renderer pour rendre le contenu plus lisible, ce qui peut être particulièrement intéressant pour le cas d'objets lourds. Cela peut être réalisé en implémentant l'interface **LoggerRenderer** et en modifiant la configuration.

```
class magazineRenderer implements LoggerRenderer {
  public function render($magazine) {
    return "{$magazine->number} : {$magazine->title}";
  }
}
```

Dans votre nouvelle classe renderer, il vous faudra écrire la méthode **render**, dont le résultat sera utilisé par log4php pour formater ces objets. À condition de ne pas oublier de l'indiquer dans la configuration à l'aide de la balise **renderer** :

```
<configuration xmlns="http://logging.apache.org/log4php/">
  <renderer renderedClass="Magazine" renderingClass="MagazineRenderer" />
<appender name="default" class="LoggerAppenderEcho" />
  <root>
    <appender_ref ref="default" />
  </root>
</configuration>
```

Cette balise requiert deux paramètres, **renderClass**, qui indique à quel classe d'objets associer le renderer, et **renderingClass**, la classe à utiliser pour formater le log de l'objet. Avec cette nouvelle configuration, la sortie précédente aura cette allure :

```
INFO - Attention nous allons logguer un objet
INFO - HS 67 : MySQL & Bases de données
```

Remarquez que, une fois que vous avez défini un renderer personnalisé pour une classe, il s'appliquera non seulement à celle-ci mais aussi à toutes ses classes enfants, tant que vous ne spécifiez pas autre chose. Il y donc héritage du renderer d'une classe vers ses enfants ;

Enfin, il est également possible de remplacer le renderer par défaut. Une fois écrit votre propre renderer comme précédemment, il suffit d'ajouter une ligne dans la configuration pour indiquer qu'il faut l'utiliser par défaut à l'aide de la balise **defaultRenderer** :

```
<configuration xmlns="http://logging.apache.org/log4php/">
  <defaultRenderer renderingClass="MyRenderer" />
  <appender name="default" class="LoggerAppenderEcho" />
  <root>
    <appender_ref ref="default" />
  </root>
</configuration>
```

Celle-ci requiert un argument, **renderingClass** qui a pour valeur le nom de la classe à utiliser pour réaliser le rendu des objets.

Pour finir

Vous voilà équipé maintenant pour logguer tout ce que bon vous semble. La configuration de log4php vous donne un contrôle entier sur ce que vous voudrez conserver, où, quand et comment, sans plus intervenir dans le code, du moins si vous n'avez pas hésité à utiliser ces fonctions au cours du développement... ■

ABUS, UN AUTRE BUS LIGHT

par Thierry GAYET

Alors que les middlewares Linux montent en puissance, celui-ci s'est quelque peu complexifié au niveau des échanges. Les IPC ont longtemps régné en maître au même titre que corba, XML-RPC ou de SOAP. Le monde des bus logiciels étant déjà bien rempli coté bus avec DBUS ou ZMQ, abus a donc été développé avec le soucis d'avoir une faible empreinte mémoire, sans démon central et disposant de RPC.

1 Préambule

Il existe depuis les premiers *nix des IPC (**I**nter**P**rocess **C**ommunications) natifs permettant à des processus concurrents, locaux ou bien distants de communiquer ou de se synchroniser entre eux.

Ces IPC peuvent être regroupées en 3 catégories : les échanges de données, les synchronisations et les échanges incluant des synchros.

1.1 Échange de données

L'échange de données via IPC est l'utilisation principale entre deux ou plusieurs processus. Dans ce mode, les processus voulant envoyer des informations écrivent dans un (ou plusieurs) fichier(s). Les processus souhaitant recevoir ces informations se positionnent aux « bons » emplacements dans un fichier et les lisent. Ce type d'échange est possible entre des processus, en utilisant le système de fichiers locales ou distants, en utilisant un système de fichiers distribués tel que NFS (aka *Network FileSystem*) (Fig. 1).

La mémoire (principale) d'un système peut aussi être utilisée pour des échanges de données.

Suivant le type de processus, les outils utilisés ne sont pas les mêmes :

- Dans le cas des processus « classiques », l'espace mémoire du processus n'est pas partagé. On utilise alors des mécanismes de mémoire partagée, comme les segments de mémoire partagée pour Unix.
- Dans le cas des processus légers l'espace mémoire est partagé, la mémoire peut donc être utilisée directement.

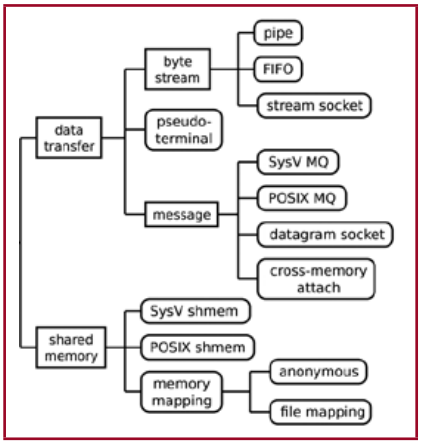


Fig. 1

Dans les deux cas, les échanges sont réalisés en plaçant les données dans des variables partagées par les processus.

Quelle que soit la méthode utilisée pour partager des données, ce type de communication pose le problème avec les sections critiques lorsqu'un processus accèdent aux données partagées. En effet si deux processus accèdent « en même temps » à une ressource, il peut se produire les problèmes suivants :

- la cohérence des données n'est plus assurée à cause d'un mélange entre le lecteur et l'écrivain ;
- un des processus reste bloqué dans l'attente de la libération de la ressource ;
- un des processus concernés plantent avec une erreur de segmentation (segfault) ;

En utilisant des fichiers, on a généralement le deuxième ou le troisième cas. Si on le prévoit, le programme peut attendre (10 millisecondes, 1 seconde etc.) et reprendre plus tard l'accès aux données. Cela dit, cette solution n'est pas toujours possible en réseau, car les fichiers ne sont pas toujours libérés correctement (par le programme, par le système d'exploitation etc.)

En utilisant la mémoire, on a plutôt le premier cas si on ne gère rien de particulier. Cela dit, on peut prévoir des synchronisations par lectures/écritures exclusives et mettre simplement

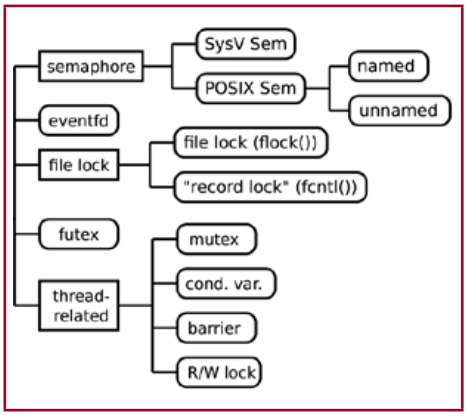


Fig. 2

le programme en attente en cas de conflit. Dans tous les cas, le partage de données en mémoire n'est possible que sur un seul et même ordinateur.

1.2 Synchronisation

Les mécanismes de synchronisation sont utilisés pour résoudre les problèmes d'accès concurrents à certaines sections critiques et plus généralement pour bloquer ou débloquer des processus selon certaines conditions (Fig. 2).

Les verrous permettent donc de protéger tout ou une partie d'un fichier en la verrouillant. Ces blocages peuvent être réalisés soit pour des opérations de lecture, d'écriture, ou bien les deux.

Un exemple de verrous est les sémaphores qui sont un mécanisme de type particulier permettant de limiter l'accès concurrent à une section critique entre processus. Pour ce faire, deux fonctions sont utilisées : P (décrémente un compteur) et V (incrémente un compteur).

Un autre type est les signaux qui sont à l'origine destinés a envoyer un événement à un processus donné comme la demande d'une interruption via le signal **SIGSEGV** :

La commande **ipcs** pemet de donner une synthèse sur l'usage des IPC sur un système donné :

```
$ ipcs
------ Segment de mémoire partagée --------
clé      shmid     propriétaire perms      octets    nattch     états
0x63637069 0         root       666        65536     0
0x00000000 32769     tgayet     700        184320    2          dest
0x00000000 65538     tgayet     700        8110080   2          dest
------ Tableaux de sémaphores --------
clé      semid     propriétaire perms      nsems
0x002fa327 0         root       666        2
0x63637069 32769     root       666        3
------ Queues de messages --------
clé      msqid     propriétaire perms      octets utilisés messages
```

1.3 Échange de données et synchronisation

Ces outils regroupent les possibilités des deux autres et sont plus simples d'utilisation. L'idée de ce genre d'outil est de communiquer en utilisant le principe des files, les processus voulant envoyer des informations les placent dans la file ; ceux voulant les recevoir les récupèrent dans cette même file. Les opérations d'écriture et de lecture dans la file sont bloquantes et permettent donc la synchronisation.

Ce principe est utilisé par les files d'attente de message (*message queue*) sous Unix, par les sockets Unix ou Internet, par les tubes, nommés ou non et par la transmission de messages (*message passing*).

1.4 Bilan des IPC ?

Bien qu'intéressant, ces mécanismes basiques ne disposent pas de toute la souplesse d'un bus logiciel récent permettant de faire l'abstraction du système ou bien des processus même. En effet, lorsque l'on veut communiquer avec un IPC, il faut soit connaître son descripteur de fichiers, soit connaître sa clef, soit enfin connaître son nom.

En effet, il est intéressant de pouvoir disposer de plusieurs fonctionnalités comme :

- sécurité globale au système de bus ;
- envoie de messages et pas forcément de flux d'octets ;
- communication locale ou distante ;
- utilisation de fonctions pour une API déporté ;
- communication avec un ou plusieurs processus ;
- échanges synchrones ou asynchrones ;
- Protection par type de données.

La communauté possède une offre pléthorique d'alternatives aux IPC natives comme CORBA, DCE, DCOM, DCOP, XML-RPC, SOAP, MBUS, *Internet Communications Engine* (ICE) et bien d'autres encore. Certains sont locaux à un système et d'autres permettent d'interagir avec d'autres à travers un réseau.

Bien évidement, il n'existe pas de solutions miracles mais plutôt des solutions plus ou moins adaptés à certaines solutions que d'autres.

2 Quels autres bus logiciels ?

Avant de rentrer dans le détail d'ABUS, je vais présenter rapidement d'autres bus existant dont Zeromq, openbinder utilisé sous Google/Android et enfin le non moins célèbre Dbus. Cela permettra de pouvoir comparer. Abus n'a jamais voulu prétendre de tout révolutionner mais bien d'offrir une alternative à faible empreinte mémoire tout comme l'a zmq mais en gardant la souplesse de l'API de Dbus.

2.1 Zeromq (aka 0MQ)

Zeromq est un des bus logiciel ayant une faible empreinte mémoire. Son fonctionnement est assez simple et possède une API complète pour un usage générique. De plus, il a un ensemble de binding pour les principaux langages :

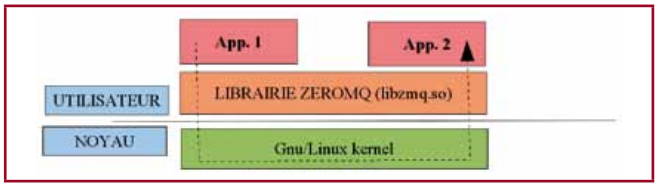


Fig. 3 : Architecture de 0mq native

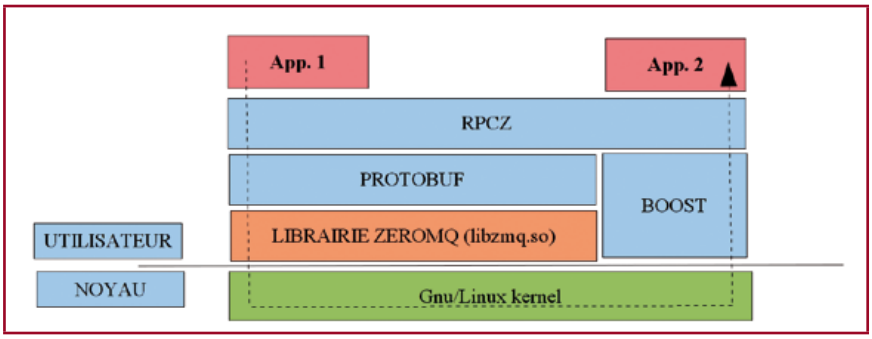


Fig. 4 : Architecture de 0mq avec la surcouche incluant le support des RPC

Dans le domaine des bus light, 0mq est assez populaire mais ne disposent pas de RPC. Pour remédier à cela, il est cependant possible de rajouter plusieurs briques externes en complément : **rpcz**, **protobuf** et **boost** (**libboost-program-options** et **boost-thread**) (Fig. 4).

Résumé des performances :

- Deux changements de contexte nécessaire pour l'envoie d'un message App.1 -> App.2 et deux autres pour le code de retour ;
- N'utilise pas de démon central ;

2.2 OPENBINDER

Openbinder est similaire à kbus (http://kbus.readthedocs.org/en/latest/specification.html#the-basics) car il est localisé dans le noyau GNU/Linux. C'est est un mécanisme de RPC basé sur des IPC, similaire à COM sous Windows. A l'origine, il a été développé pour BeOS puis Palm et enfin le portage opensource de BeOS, à savoir Haiku. Il est actuellement utilisé dans le *middleware* Google/Android. Depuis la version 3.3 du noyau GNU/Linux, le code issue d'Android a été fusionné dans **drivers/staging/android/**.

Dans l'espace noyau, **open-binder** est un driver de caractère accessible via **/dev/binder**. Il est utilisé pour transmettre des données via des appels **ioctl()**. Il autorise un gestionnaire de contenu.

La communication inter-processus (IPC) peut entraîner des trous de sécurité, c'est pour cela qu'Android à son propre IPC, le Binder, et que la communication inter-processus n'est pas laissé aux développeurs d'application. De plus, avoir un système IPC centralisé permet une maintenance plus facile et une correction des problèmes de sécurités générales.

Dans Android chaque application est lancée dans un processus séparé. Ces différents processus ont besoin de communiquer l'IPC Binder. Il permet à plusieurs processus de partager des données, de communiquer entre eux en utilisant le partage mémoire via le driver **ashmem**. Cette technique permet des performances accrues par rapports aux recopies de zones de mémoire ou bien de la sérialisation.

Les problèmes de concurrence, lorsque plusieurs processus essayent d'accéder en même temps à une « même zone mémoire » (au même objet java) sont gérés par le Binder. Tous les appels sont synchronisés entre les processus (Fig. 5).

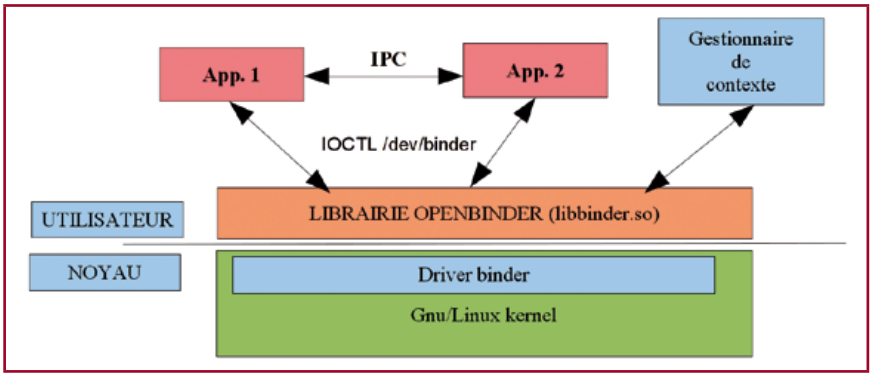


Fig. 5 : Architecture d'OpenBinder utilisé dans le framework Android

L'application A récupère une référence vers le Service B à l'aide du gestionnaire de contexte. Ce gestionnaire peut être comparé à un DNS car il permet de récupérer à l'aide d'un nom, une référence vers un objet java. Cette méthode est similaire au registre dans RMI (*Remote Method Invocation*). Si on veut partager des objets, il faut donc s'être au préalable enregistré auprès de ce gestionnaire de contexte.

Une fois la référence vers le service B récupéré, la méthode **hello()** du service B est appelé par l'application A. Le binder intercepte cet appel et, à l'aide d'un des threads libres présent dans sa *thread pool* (piscine de thread ou réservoir de threads), il va exécuter la méthode sur le service B.

Résumé des performances :

- Deux changements de contexte nécessaires pour l'envoie d'un message App.1 -> App.2 et deux autres pour le code de retour. De plus, ne pas sous-entendre les échanges avec le gestionnaire de contexte ;
- N'utilise pas de démon central mais possède un gestionnaire de contexte en espace utilisateur pour la résolution de nom ;

2.3 DBUS

Fort du constat de voir que certain ORBs Corba ou d'autres observait un manque de performances, DBUS à vu le jour comme une alternative open-source avec comme mission d'offrir une API

complète pour l'échange de messages plus ou moins complexes avec le maximum de performances.

Même si le coté performance n'est pas forcément au rendez-vous, il a su s'imposer au fil des années sur des composants comme le « NetworkManager », CUPS et bien d'autres.

DBUS procède donc par l'envoie de messages qu'il complète avec une entête pour les méta-données et un payload pour les données. Le format des messages est binaire, typé, complètement aligné et simple.

Il est articulé autour d'un daemon central (**dbus-daemon**) et d'une librairie de wrapping (**libdbus**).

Après enregistrement d'une première application A et d'une seconde application B, les deux peuvent échanger des messages via le daemon servant de routeur de messages inter-processus.

La communication intrinsèque se faisant via des sockets UNIX ou INET selon le mode de compilation du BUS. **AF_UNIX** permettra une communication locale à un host alors que **AF_INET** permettra l'échange de données inter-devices.

Cependant, comme tout IPC qui se respecte, un passage obligé par le noyau est nécessaire, même si des échanges point à point est possible. Les échanges par sockets aboutissent à des changements de contextes entre l'espace utilisateur et l'espace noyau. Ainsi, un échange de messages nécessite 4 changements de contexte entre deux processus et autant pour le retour. Une fréquence élevée au niveau des échanges de messages entre processus peuvent donc avoir un impact global sur le système (Fig. 6).

La communication par un bus logiciel se fait en plusieurs étapes :

1. enregistrement des services (souvent un par binaire mais pouvant être démultiplié) : cette étape permet de se déclarer auprès du daemon par rapport à un service donné ;
2. communication par message ou signal (broadcast) ;

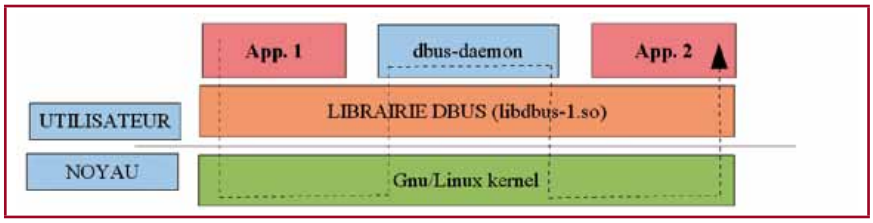


Fig. 6 : Architecture de DBUS

3. dés-enregistrement au bus : cette étape nous permet de nous enlevé de la liste des programmes

Ainsi, lorsque le programme A veut envoyer une message au programme B, il le fait par l'intermédiaire du daemon. En effet, le programme A émet le message au daemon, puis ce dernier le renvoie au programme B, puisque ce dernier a une connaissance des programmes enregistrés et apte à communiquer.

Grossièrement, ce daemon possède une table d'association du file descripteur de la socket utilisée avec un programme. Cette connaissance lui permet de savoir à qui envoyer un message car en interne, un message est envoyé en XML tel un paquet IP avec une source et un destinataire.

DBUS est utilisé sur les desktop mais aussi sur certains devices comme des smartphones, tablettes ou settopbox.

Cependant, malgré tout, DBUS qui se voulait le successeur de CORBA en version plus light, n'a pas vraiment tenu ses promesses et accuse aujourd'hui un manque certain de performances ce qui lui a lancé le développent light qui a aboutit à ABUS.

Résumé des performances :

- Quatre changements de contexte pour l'envoie d'un message App.1 -> App.2 et autant pour le retour ;
- Utilise un démon central jouant le rôle de routeur de paquets.

3 ABUS

Abus est né au sein de la société AVIWEST localisée à Rennes. Créée en 2008 par cinq ingénieurs de Thomson, cette société a mis au point une technologie qui permet aux équipes de télévision de terrain de pouvoir transmettre des vidéos de terrain en direct en 3G/4G sans recourir à des moyens lourds comme les liaisons satellites.

Le développement à donc été lancé fin 2010 avec comme objectif d'être adapté au monde de l'embarqué, c'est à dire sans démon central, avec une faible empreinte mémoire, pouvant communiquer en synchrone/asynchrone. Le tout devant posséder des RPC et une utilisation simple de par son API.

3.1 Architecture

L'architecture d'abus se calque que le modèle utilisé par Dbus avec une librairie de wrapping intermédiaire mais

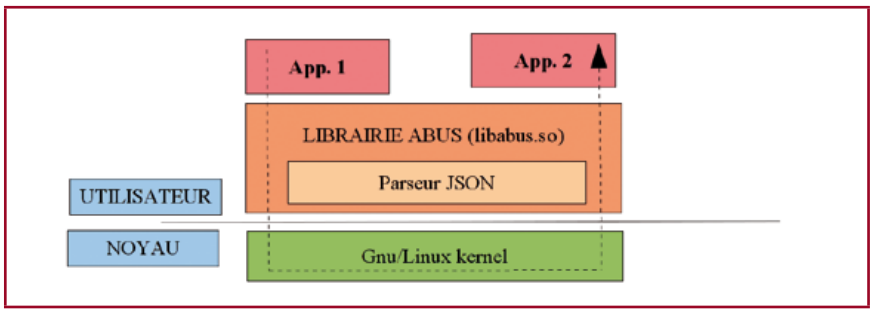


Fig. 7 : Architecture de abus

sans daemon. Abus est implémenté en local sur des sockets BSD **AF_UNIX**. Il y a donc deux fois moins de changement de contexte : voir Figure 7 page précédente.

Les échanges de messages se font dans Abus suivant la syntaxe Json ce qui explique que abus intègre un parser en interne.

Abus se base sur une librairie offrant les différents mécanismes de communication basés sur des sockets Unix (donc local au système).

La visualisation des services connectés peut être visualisé en consultant le répertoire **/tmp/abus** :

```
$ ls -al /tmp/abus/
total 0
drwxr-xr-x   2 root  root  460 Feb 22 14:53 .
drwxrwxrwt  13 root  root  700 Feb 22 15:59 ..
srwxr-xr-x   1 root  root    0 Feb 22 14:53 _654
srwxr-xr-x   1 root  root    0 Feb 22 14:53 _663
lrwxrwxrwx   1 root  root    4 Feb 22 14:53 servicesvc -> _654
lrwxrwxrwx   1 root  root    4 Feb 22 14:53 demo -> _663
```

Pour surveiller les services en temps réel, la commande **watch** est d'une grande utilité :

```
$ watch ls /tmp/abus/
```

```
Every 2,0s: ls /tmp/abus/

_29828
examplesvc
```

Fig. 8

Tout comme Dbus, abus dispose d'un outils nommé **abus-send** permettant d'envoyer des commandes sur le bus et de simuler un client.

Il n'y a actuellement pas de mécanisme de monitoring des échanges sur le bus global tel que l'outil **dbus-monitor**. A noter l'existence d'une variable d'environnement permettant d'activer les traces sur abus :

```
export ABUS_MSG_VERBOSE=1
```

3.2 Obtenir abus

Abus ne fournit aucune version runtime pré-compilé pour une architecture.

La première étape est donc la récupération du code soit sous forme de d'archive ou via le *repository* subversion :

```
$ wget http://abus.googlecode.com/files/abus-0.1.tar.gz
$ tar zxvf abus-0.1.tar.gz
$ cd abus-0.1/
```

ou bien depuis le SVN de googlecode :

```
$ svn checkout http://abus.googlecode.com/svn/trunk/ abus-read-only
$ cd abus-read-only/
```

Au moment de la rédaction, la révision courante est la « 124 ».

Avant de lancer le *bootstrap* **autotools**, listons les fichiers pour obtenir un différentiel plus tard :

```
$ ls
abus  abus.pc.in  AUTHORS
autogen.sh  configure.ac  doc
examples  LICENSE  m4  Makefile.am
NEWS  README  test  tools
```

Une fois le code obtenu, il faudra lancer le scripts de bootstrap **autogen.sh** qui servira à générer les fichiers autotools et notamment le script configure. Cette étape revient à lancer **./autoreconf --force --install** :

```
$ ./autogen.sh
autoreconf2.50: Entering directory `.'
autoreconf2.50: configure.ac: not using Gettext
autoreconf2.50: running: aclocal --force -I m4
autoreconf2.50: configure.ac: tracing
autoreconf2.50: running: libtoolize --install --copy --force
libtoolize: putting auxiliary files in `.'.
libtoolize: copying file `./config.guess'
libtoolize: copying file `./config.sub'
libtoolize: copying file `./install-sh'
libtoolize: copying file `./ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIR, `m4'.
libtoolize: copying file `m4/libtool.m4'
libtoolize: copying file `m4/ltoptions.m4'
libtoolize: copying file `m4/ltsugar.m4'
libtoolize: copying file `m4/ltversion.m4'
libtoolize: copying file `m4/lt~obsolete.m4'
autoreconf2.50: running: /usr/bin/autoconf --force
autoreconf2.50: running: /usr/bin/autoheader --force
autoreconf2.50: running: automake --add-missing --copy --force-missing
configure.ac:32: installing `./compile'
configure.ac:12: installing `./missing'
abus/Makefile.am: installing `./depcomp'
autoreconf2.50: Leaving directory `.'
```

Après cette étape on remarquera la création d'un certain nombre de fichiers :

```
$ ls
abus   aclocal.m4  autom4te.cache
config.sub    depcomp   install-sh  m4
missing  test abus_config.h.in  AUTHORS
compile  configure  doc LICENSE  Makefile.am
NEWS  tools  abus.pc.in  autogen.sh
config.guess  configure.ac  examples
ltmain.sh   Makefile.in  README
```

Consultons les paramètres du script **./configure** pour abus :

```
$ ./configure -help
[...]
  --enable-fcgi      Enable FastCGI gateway [default=no]
  --enable-test      Enable the test unitary support [default=no]
  --enable-examples  Enable the examples [default=no]
[...]
```

On voit donc apparaître trois paramètres spécifiques au package :

1. **enable-fcgi** : permet de compiler un plugin Abus pour Lighttpd ; nécessite l'installation du package **lib-fcgidev** ;
2. **enable-test** : permet l'activation des tests unitaires ; nécessite l'installation du package **google/test** ;
3. **enable-examples** : permet la compilation des exemples ; ne requiert pas de dépendances spécifiques.

Abus étant un projet récent, il n'existe malheureusement pas de package Debian, Fedora ou autre.

Le projet livre librement les sources sous licence LGPL ce qui autorise le linkage dynamique sans contamination pouvant entraîner une divulgation du code source : http://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_limit%C3%A9e_GNU.

3.3 Installation des dépendances

3.3.1 Google/test

Avant de pouvoir compiler et utiliser Abus, il peut être nécessaire d'installer certaines dépendances comme la librairie **libgtest** ou la **libfcgi**.

Abus disposant d'une batterie de tests unitaires basés sur le framework **google/test** il faudra au préalable l'installer.

Google test n'étant pas adapté à 100 % aux **autotools**, je conseille d'utiliser une version déjà patché depuis mon site web perso :

```
$ wget www.nextinnovation.org/tarballs/gtest-1.6.0.tar.gz
$ tar zxvf gtest-1.6.0.tar.gz && cd gtest-1.6.0
$ autoreconf -i --force
$ ./configure
$ make
$ sudo make install
```

Une fois installé, les tests unitaires pourrons être lancés.

3.3.2 fcgi

Si l'on souhaite connecter le Abus en distant via un serveur HTTP Lighttpd il faudra ajouter une nouvelle dépendance :

```
$ sudo apt-get install libfcgi-dev libfcgi0ldbl
```

Nous verrons cette utilisation plus tard.

3.4 Compilation d'Abus

Une fois les dépendances installées, il est possible de configurer le package Abus.

Exemple d'activation incluant le module **fcgi** pour Lighttpd, le support des tests unitaires ainsi que les exemples :

```
$ ./configure --enable-fcgi --enable-test --enable-examples
[...]
checking for FastCGI support... yes
checking for FCGI_Accept in -lfcgi... yes
checking whether to enable the test support... yes
checking for pkg-config... /usr/bin/pkg-config
checking pkg-config is at least version 0.9.0... yes
checking for GTEST... yes
checking whether to build the examples... yes
configure: creating ./config.status
config.status: creating m4/Makefile
config.status: creating abus/Makefile
config.status: creating abus/libjson/Makefile
config.status: creating abus/hashtab/Makefile
config.status: creating examples/Makefile
config.status: creating test/Makefile
config.status: creating doc/Makefile
config.status: creating doc/Doxyfile
config.status: creating tools/Makefile
config.status: creating Makefile
config.status: creating abus.pc
config.status: creating abus_config.h
config.status: executing depfiles commands
config.status: executing libtool commands
```

Cette étape passé, il est possible de lancer la génération d'Abus :

```
$ make
Making all in m4
Making all in abus
Making all in hashtab
  CC     hashtab.lo
  CC     recycle.lo
[...]
  CC     example-json-config.o
  CCLD   example-json-config
Making all in test
Making all in doc
```

Le mode silencieux est activé par défaut au niveau du **Makefile**. Pour le passer en mode « verbose », il faudra lui passer le paramètre **V=1** :

Pour la phase d'installation, un simple **make install** précédé d'un **sudo** suffit :

```
$ sudo make install
Password:
Making install in m4
make[2]: Rien à faire pour " install-exec-am ".
make[2]: Rien à faire pour " install-data-am ".
Making install in abus
Making install in hashtab
[...]
```

Pour changer le répertoire de destination, le paramètre **DESTDIR** est prévu pour cela mais vaut mieux laisser abus s'installer dans son chemin par défaut.

De façon à mettre à jour le cache des librairies, actualisons le cache pour le chargeur ELF :

```
$ sudo ldconfig
```

Vérifions la mise à jour du cache :

```
$ ldconfig -p | grep -i abus
  libabus.so.1 (libc6) => /usr/local/lib/libabus.so.1
  libabus.so (libc6) => /usr/local/lib/libabus.so
```

Lançons les tests unitaires pour valider avant d'aller plus loin :

```
$ make check
Making check in m4
Making check in abus
Making check in hashtab
Making check in libjson
Making check in tools
Making check in examples
Making check in test
  CXX    abus_test-abus-tor.o
  CXX    abus_test-abus-svc.o
  CXX    abus_test-abus-attr.o
  CXX    abus_test-abus-event.o
  CXX    abus_test-abus-introspect.o
  CXX    abus_test-AbusTest.o
  CXX    abus_test-json-config.o
  CXXLD  abus-test
Running main() from gtest_main.cc
[==========] Running 36 tests from 12 test cases.
[----------] Global test environment set-up.
[----------] 14 tests from AbusReqTest
[ RUN      ] AbusReqTest.BasicSvc
##  9137 -> tmp/abus/gtestsvc:71 {"jsonrpc":"2.0","method":"gtestsvc.
sum","id":0,"params":{"a":2,"b":3}}
##  9137 <- 0006c:71 {"jsonrpc":"2.0","method":"gtestsvc.sum","id":0,"
params":{"a":2,"b":3}}
##  9137 -> 0006c:49 {"jsonrpc":"2.0","result":{"res_value":5},"id":0}
##  9137 <- :49 {"jsonrpc":"2.0","result":{"res_value":5},"id":0}
[       OK ] AbusReqTest.BasicSvc (1 ms)
(...)
[==========] 36 tests from 12 test cases ran. (2239 ms total)
[  PASSED  ] 36 tests.
PASS: abus-test
============
1 test passed
============
Making check in doc
```

Tous les 36 tests doivent passer en **RUN** et ne pas être au status **FAILLED**. Si tout est ok c'est que vous êtes prêt à utiliser abus ce qui est une bonne nouvelle.

4 Premier pas avec Abus

Une fois Abus installé, il est possible de jouer via divers code d'exemples. Cela servira aussi à valider le bon fonctionnement du bus :

```
$ cd examples/
$ ls
example-client        example-clientpp.o    example-clnt-attr.c
example-json-config    example-service.o      example-servicexx.cpp
example-svc-attr     example-svc-event.o  jquery.js          Makefile.in
example-client.c      example-clnt-array    example-clnt-attr.o
example-json-config.c  example-servicepp      example-servicexx.o
example-svc-attr.c   example-svc-poll     jquery.jsonrpc.js  rpc.sh
example-client.o      example-clnt-array.c  example-clnt-multiattr
example-json-config.o  example-servicepp.cpp  example-svc-array
example-svc-attr.o   example-svc-poll.c   lighttpd.conf
example-clientpp      example-clnt-array.o  example-clnt-multiattr.c
example-service        example-servicepp.o    example-svc-array.c
example-svc-event    example-svc-poll.o   Makefile
example-clientpp.cpp  example-clnt-attr     example-clnt-multiattr.o
example-service.c      example-servicexx      example-svc-array.o
example-svc-event.c  index.html          Makefile.am
```

Un premier test revient à démarrer un serveur en tâche de fond qui déclarera un service et ses RPC sur le bus :

```
$ ./example-service &
[1] 11644
```

Maintenant que le service est lancé en tâche de fond, on peut récupérer le nom du service déclaré par introspection :

```
$ abus-send .\*
## 14186 -> tmp/abus/examplesvc:94 {"jsonrpc":"2.0","method":"example
svc.get","id":1,"params":{"attr":[{"name":"abus.version"}]}}
## 11644 <- 001bd:94 {"jsonrpc":"2.0","method":"examplesvc.get","id":
1,"params":{"attr":[{"name":"abus.version"}]}}
## 11644 -> 001bd:66 {"jsonrpc":"2.0","result":{"abus.version":"A-Bus
0.3~svn"},"id":1}
## 14186 <- :66 {"jsonrpc":"2.0","result":{"abus.version":"A-Bus
0.3~svn"},"id":1}
services[0]:
  name=" examplesvc"
```

On trouve donc que l'exemple de service a déclaré le service **examplesvc** ce qui permet de lancer ensuite une introspection dessus :

```
$ abus-send examplesvc.\*
## 12507 -> tmp/abus/examplesvc:60 {"jsonrpc":"2.0","method":"examplesvc.*","id
":0,"params":{}}
## 11644 <- 001b9:60 {"jsonrpc":"2.0","method":"examplesvc.*","id":0,"para
ms":{}}
## 11644 -> 001b9:493 {"jsonrpc":"2.0","result":{"methods":[{"name":"sum
","flags":0,"descr":"Compute summation of two integers","fmt":"a:i:first
operand,b:i:second operand","result_fmt":"res_value:i:summation"},{"name":"mul
t","flags":0,"descr":"Compute multiplication of two integers","fmt":"a:i:first
operand,b:i:second operand","result_fmt":"res_value:i:multiplication"}],"attrs":
[{"name":"abus.version","type":"s","readonly":true,"constant":true,"descr":"Vers
ion of the A-Bus library for this service"}]},"id":0}
## 12507 <- :493 {"jsonrpc":"2.0","result":{"methods":[{"name":"sum","flags":0,
"descr":"Compute summation of two integers","fmt":"a:i:first operand,b:i:second
operand","result_fmt":"res_value:i:summation"},{"name":"mult","flags":0,"descr
":"Compute multiplication of two integers","fmt":"a:i:first operand,b:i:second
operand","result_fmt":"res_value:i:multiplication"}],"attrs":[{"name":"abus.ver
sion","type":"s","readonly":true,"constant":true,"descr":"Version of the A-Bus
library for this service"}]},"id":0}
attrs[0]:
  descr="Version of the A-Bus library for this service"
  name="abus.version"
  type="s"
  readonly=true
  constant=true
methods[0]:
  descr="Compute summation of two integers"
  fmt="a:i:first operand,b:i:second operand"
  name="sum"
  result_fmt="res_value:i:summation"
  flags=0
methods[1]:
  descr="Compute multiplication of two integers"
  fmt="a:i:first operand,b:i:second operand"
  name="mult"
  result_fmt="res_value:i:multiplication"
  flags=0
```

On voit que pour ce service, il y a deux méthodes RPC de déclarées avec la syntaxe de leur appel :

Nom de la méthode	Paramètre(s)	Retour
sum	**a** : entier ou i / **b** : entier ou i	**res_value**: entier
multi	**a** : entier ou i / **b** : entier ou i	**res_value** : entier

Testons un appel avec un client adapté en lui passant deux nombres à additionner :

```
$ ./example-client sum 4 5
## 13479 -> tmp/abus/examplesvc:73 {"jsonrpc":"2.0","method":"example
svc.sum","id":0,"params":{"a":4,"b":5}}
## 11644 <- 001bb:73 {"jsonrpc":"2.0","method":"examplesvc.sum","id":
0,"params":{"a":4,"b":5}}
## svc_sum_cb: arg=sumator cookie, ret=0, a=4, b=5, => result=9
## 11644 -> 001bb:49 {"jsonrpc":"2.0","result":{"res_
value":9},"id":0}
## 13479 <- :49 {"jsonrpc":"2.0","result":{"res_value":9},"id":0}
res_value=9
```

Le résultat est correct : 4+5 = 9 ... l'honneur est sauf :-)

Testons la même invocation, mais cette fois-ci avec la commande **abus-send** :

```
$ abus-send examplesvc.sum a=4 b=5
## 12771 -> tmp/abus/examplesvc:73 {"jsonrpc":"2.0","method":"example
svc.sum","id":0,"params":{"a":4,"b":5}}
## 11644 <- 001ba:73 {"jsonrpc":"2.0","method":"examplesvc.sum","id":
0,"params":{"a":4,"b":5}}
## svc_sum_cb: arg=sumator cookie, ret=0, a=4, b=5, => result=9
## 11644 -> 001ba:49 {"jsonrpc":"2.0","result":{"res_
value":9},"id":0}
## 12771 <- :49 {"jsonrpc":"2.0","result":{"res_value":9},"id":0}
res_value=9
```

Heureusement on trouve la même chose !

5 Présentation de l'API d'ABUS

L'API d'Abus est décrite dans le header **abus.h**. Elle regroupe l'ensemble des fonctions utilisables :

`abus_t* abus_init(const abus_conf_t *conf);` `int abus_cleanup(abus_t *abus);`	**Initialisation / libération du contexte abus**
`int abus_decl_method(abus_t *abus, const char *service_name, const char *method_name, abus_callback_t method_callback, int flags, void *arg, const char *descr, const char *fmt, const char *result_fmt);` `int abus_undecl_method(abus_t *abus, const char *service_name, const char *method_name);`	**Ajout / Suppression d'une association d'une RPC (via une callback) à un service**

`int abus_decl_attr_int(abus_t *abus, const char *service_name, const char *attr_name, int *val, int flags, const char *descr);`
`int abus_decl_attr_llint(abus_t *abus, const char *service_name, const char *attr_name, long long *val, int flags, const char *descr);`
`int abus_decl_attr_bool(abus_t *abus, const char *service_name, const char *attr_name, bool *val, int flags, const char *descr);`
`int abus_decl_attr_double(abus_t *abus, const char *service_name, const char *attr_name, double *val, int flags, const char *descr);`
`int abus_decl_attr_str(abus_t *abus, const char *service_name, const char *attr_name, char *val, size_t n, int flags, const char *descr);`
`int abus_undecl_attr(abus_t *abus, const char *service_name, const char *attr_name);`
`int abus_attr_changed(abus_t *abus, const char *service_name, const char *attr_name);`
`int abus_append_attr(abus_t *abus, json_rpc_t *json_rpc, const char *service_name, const char *attr_name);`
	Fonctions orientées service

Fonctions orientées client

```
int abus_attr_get_int(abus_t *abus, const char *service_name, const char *attr_name, int *val, int timeout);

int abus_attr_get_llint(abus_t *abus, const char *service_name, const char *attr_name, long long *val, int timeout);

int abus_attr_get_bool(abus_t *abus, const char *service_name, const char *attr_name, bool *val, int timeout);

int abus_attr_get_double(abus_t *abus, const char *service_name, const char *attr_name, double *val, int timeout);

int abus_attr_get_str(abus_t *abus, const char *service_name, const char *attr_name, char *val, size_t n, int timeout);

int abus_attr_set_int(abus_t *abus, const char *service_name, const char *attr_name, int val, int timeout);

int abus_attr_set_llint(abus_t *abus, const char *service_name, const char *attr_name, long long val, int timeout);

int abus_attr_set_bool(abus_t *abus, const char *service_name, const char *attr_name, bool val, int timeout);

int abus_attr_set_double(abus_t *abus, const char *service_name, const char *attr_name, double val, int timeout);

int abus_attr_set_str(abus_t *abus, const char *service_name, const char *attr_name, const char *val, int timeout);

int abus_attr_subscribe_onchange(abus_t *abus, const char *service_name, const char *attr_name, abus_callback_t callback, int flags, void *arg, int timeout);

int abus_attr_unsubscribe_onchange(abus_t *abus, const char *service_name, const char *attr_name, abus_callback_t callback, void *arg, int timeout);
```

Fonctions de gestion d'évènements

```
int abus_decl_event(abus_t *abus, const char *service_name, const char *event_name, const char *descr, const char *fmt);

int abus_undecl_event(abus_t *abus, const char *service_name, const char *event_name);

json_rpc_t* abus_request_event_init(abus_t *abus, const char *service_name, const char *event_name);

int abus_request_event_publish(abus_t *abus, json_rpc_t *json_rpc, int flags);

int abus_request_event_cleanup(abus_t *abus, json_rpc_t *json_rpc);

int abus_event_subscribe(abus_t *abus, const char *service_name, const char *event_name, abus_callback_t callback, int flags, void *arg, int timeout);

int abus_event_unsubscribe(abus_t *abus, const char *service_name, const char *event_name, abus_callback_t callback, void *arg, int timeout);
```

Liste des fonctions synchrones

```
json_rpc_t* abus_request_method_init(abus_t *abus, const char *service_name, const char *method_name);

int abus_request_method_invoke(abus_t *abus, json_rpc_t* json_rpc, int flags, int timeout);

int abus_request_method_cleanup(abus_t *abus, json_rpc_t *json_rpc);
```

Liste des fonctions asynchrones

```
int abus_request_method_invoke_async(abus_t *abus, json_rpc_t
*json_rpc, int timeout, abus_callback_t callback, int flags,
void *arg);

int abus_request_method_wait_async(abus_t *abus, json_rpc_t
*json_rpc, int timeout);

int abus_request_method_cancel_async(abus_t *abus, json_rpc_t
*json_rpc);
```

Les méthodes sont typées au niveau des paramètres. Les types suivants sont supportés :

- **i** : entier
- **s** : chaîne de caractères
- **b** : booléen
- **l** : **long int**

Pour chaque exemple, je vais décrire le code synchrone d'un service et d'un client.

5.1 Exemple de service et client simple

Ce premier exemple va détailler le service utilisé préalablement pour tester abus après son installation. Le service déclare deux fonctions RPC **sum** et **mult** revenant à faire l'addition ou la multiplication de deux nombres passés en paramètre :

Code source du service :

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#include "abus.h"

static void svc_sum_cb(json_rpc_t *json_rpc, void *arg)
{
        int a, b;
        int ret;

       // Récupération des deux valeurs " a " et " b " à additionner

        ret  = json_rpc_get_int(json_rpc, "a", &a);
        if (ret == 0)
                ret = json_rpc_get_int(json_rpc, "b", &b);


        printf("## %s: arg=%s, ret=%d, a=%d, b=%d, => result=%d\n", __func__,
(const char*)arg, ret, a, b, a+b);

       // Renvoie du résultat " a+b "
        if (ret)
                json_rpc_set_error(json_rpc, ret, NULL);
        else
                json_rpc_append_int(json_rpc, "res_value", a+b);
}

static void svc_mult_cb(json_rpc_t *json_rpc, void *arg)
{
        int a, b;
        int ret;

       // Récupération des deux valeurs " a " et " b " à multiplier
        ret  = json_rpc_get_int(json_rpc, "a", &a);
        if (ret == 0)
                ret = json_rpc_get_int(json_rpc, "b", &b);


        printf("## %s: arg=%s, ret=%d, a=%d, b=%d, => result=%d\n", __func__,
                                        (const char*)arg, ret, a, b, a*b);

       // Renvoie du résultat " a*b "
        if (ret)
                json_rpc_set_error(json_rpc, ret, NULL);
        else
                json_rpc_append_int(json_rpc, "res_value", a*b);
}

int main(int argc, char **argv)
{
        abus_t *abus;
        int ret;

        abus = abus_init(NULL);

        ret = abus_decl_method(abus, "examplesvc", "sum", &svc_sum_cb,
                                        ABUS_RPC_FLAG_NONE,
                                       "sumator cookie",
                                       "Compute summation of two integers",
                                       "a:i:first operand,b:i:second operand",
                                       "res_value:i:summation");
        if (ret != 0) {
                abus_cleanup(abus);
                return EXIT_FAILURE;
        }

        ret = abus_decl_method(abus, "examplesvc", "mult", &svc_mult_cb,
                                       ABUS_RPC_FLAG_NONE,
                                       "multiply cookie",
                                       "Compute multiplication of two
integers",
                                       "a:i:first operand,b:i:second operand",
                                       "res_value:i:multiplication");
        if (ret != 0) {
                abus_cleanup(abus);
                return EXIT_FAILURE;
        }

        /* do other stuff */
        sleep(10000);

        abus_cleanup(abus);

        return EXIT_SUCCESS;
}
```

Code source du client associé :

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include "abus.h"

#define RPC_TIMEOUT 1000 /* ms */

int main(int argc, char **argv)
{
        abus_t *abus;
        json_rpc_t *json_rpc;
        int ret, res_value;
        const char *service_name = "examplesvc";

        if (argc < 4) {
                printf("usage: %s METHOD firstvalue secondvalue\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        abus = abus_init(NULL);

        /* La méthode est donné " sum " ou " mult " */
        json_rpc = abus_request_method_init(abus, service_name, argv[1]);
        if (!json_rpc)
                exit(EXIT_FAILURE);

        // passage des deux paramètres " a " et " b "
        json_rpc_append_int(json_rpc, "a", atoi(argv[2]));
        json_rpc_append_int(json_rpc, "b", atoi(argv[3]));

        ret = abus_request_method_invoke(abus, json_rpc, ABUS_RPC_FLAG_NONE,
RPC_TIMEOUT);
        if (ret != 0) {
                printf("RPC failed with error %d\n", ret);
                exit(EXIT_FAILURE);
         }

        // Récupération du résultat
         ret = json_rpc_get_int(json_rpc, "res_value", &res_value);

         if (ret == 0)
                printf("res_value=%d\n", res_value);
         else
                printf("No result? error %d\n", ret);

         abus_request_method_cleanup(abus, json_rpc);
         abus_cleanup(abus);

         return EXIT_SUCCESS;
}
```

5.2 Exemple de passage d'une structure de données

Ce second exemple détaille le passage d'un ensemble de paramètres, regroupés dans un conteneur :

Code source du service :

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#include <abus.h>
#include <json.h>

static void svc_array_sqr_cb(json_rpc_t *json_rpc, void *arg)
{
        int k;
        int ret, count, i;
        int *ary = NULL;

        ret = json_rpc_get_int(json_rpc, "k", &k);
        if (ret != 0) {
                json_rpc_set_error(json_rpc, ret, NULL);
                return;
        }

        count = json_rpc_get_array_count(json_rpc, "my_array");
        if (count >= 0)
        {
                /* first put all the values in an array, in order to not mix
                   json_rpc_get's and json_rpc_append's for readability sake
                 */
                ary = malloc(count*sizeof(int));

                for (i = 0; i<count; i++)
                {
                        /* Aim at i-th element within array "my_array" */
                        ret = json_rpc_get_point_at(json_rpc, "my_array", i);
                          if (ret != 0)
                                  break;

                          /* from that dictionary, get parameter "a"
                        * Rem: expects all array elements to contain at least
a param "a"
                        */
                       ret = json_rpc_get_int(json_rpc, "a", &ary[i]);
                       if (ret != 0)
                              break;
                }
        }

        printf("## %s: arg=%s, ret=%d, k=%d, array count=%d\n", __func__, (const
char*)arg, ret, k, count);

        if (ret) {
                json_rpc_set_error(json_rpc, ret, NULL);
        } else {

                json_rpc_append_int(json_rpc, "res_k", k);

                /* begin the array */
                json_rpc_append_args(json_rpc,
                                                JSON_KEY, "res_array", -1,
                                                JSON_ARRAY_BEGIN,
                                                -1);

                for (i = 0; i<count; i++)
    {
                        /* each array element *must* be an "OBJECT", i.e. a
dictonary */
                        json_rpc_append_args(json_rpc, JSON_OBJECT_BEGIN, -1);

                        json_rpc_append_int(json_rpc, "res_a", ary[i]*ary[i]);
                        /* more stuff may be appended in there */

                        json_rpc_append_args(json_rpc, JSON_OBJECT_END, -1);
                }

                /* end the array */
                json_rpc_append_args(json_rpc, JSON_ARRAY_END, -1);
        }
        if (ary)
                free(ary);
}

int main(int argc, char **argv)
{
        abus_t *abus;
        int ret;

        abus = abus_init(NULL);
```

```
        ret = abus_decl_method(abus, "examplearraysvc", "sqr", &svc_array_sqr_cb,
                                     ABUS_RPC_FLAG_NONE,
                                     "square cookie",
                                     "Compute square value of all the elements
of an array. Serves as an example of how to deal with array in A-Bus",
                                     "k:i:some contant,my_array:(a:i:value to
be squared,arg_index:i:index of arg for demo):array of stuff",
                                     "res_k:i:same contant,res_
array:(res_a:i:squared value):array of squared stuff");
        if (ret != 0) {
                abus_cleanup(abus);
                return EXIT_FAILURE;
        }

        /* do other stuff */
        sleep(10000);

        abus_cleanup(abus);

        return EXIT_SUCCESS;
}
```

Code source du client associé :

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include <abus.h>
#include <json.h>

#define RPC_TIMEOUT 1000 /* ms */

int main(int argc, char **argv)
{
        abus_t *abus;
        json_rpc_t *json_rpc;
        int count, i, ret, res_value;
        const char *service_name = "examplearraysvc";

        if (argc < 4) {
                printf("usage: %s METHOD k values...\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        abus = abus_init(NULL);

        /* method name is taken from command line */
        json_rpc = abus_request_method_init(abus, service_name, argv[1]);
        if (!json_rpc)
                exit(EXIT_FAILURE);

        /* pass 2 parameters: "k" and "my_array" */
        json_rpc_append_int(json_rpc, "k", atoi(argv[2]));

        /* begin the array */
        json_rpc_append_args(json_rpc,
                                     JSON_KEY, "my_array", -1,
                                     JSON_ARRAY_BEGIN,
                                     -1);

        for (i = 3; i<argc; i++)
        {
                /* each array element *must* be an "OBJECT", i.e. a dictonary */
                json_rpc_append_args(json_rpc, JSON_OBJECT_BEGIN, -1);

                json_rpc_append_int(json_rpc, "a", atoi(argv[i]));
                /* more stuff may be appended in there */
                json_rpc_append_int(json_rpc, "arg_index", i);

                  json_rpc_append_args(json_rpc, JSON_OBJECT_END, -1);
         }
```

```
        /* end the array */
        json_rpc_append_args(json_rpc, JSON_ARRAY_END, -1);


        ret = abus_request_method_invoke(abus, json_rpc, ABUS_RPC_
FLAG_NONE, RPC_TIMEOUT);
        if (ret != 0) {
                printf("RPC failed with error %d\n", ret);
                exit(EXIT_FAILURE);
        }

        count = json_rpc_get_array_count(json_rpc, "res_array");
        if (count < 0) {
                printf("No result? error %d\n", count);
                exit(EXIT_FAILURE);
        }

        ret = json_rpc_get_int(json_rpc, "res_k", &res_value);
        if (ret == 0)
                printf("res_k=%d\n", res_value);
        else
                printf("No result? error %d\n", ret);

        for (i = 0; i<count; i++)
       {
                /* Aim at i-th element within array "res_array" */
                json_rpc_get_point_at(json_rpc, "res_array", i);

                printf("res_array[%d]\n", i);

                ret = json_rpc_get_int(json_rpc, "res_a", &res_value);
                if (ret == 0)
                        printf("\tres_a=%d\n", res_value);
                else
                        printf("\tNo result? error %d\n", ret);
        }

        /* Aim back out of array */
        json_rpc_get_point_at(json_rpc, NULL, 0);

        ret = json_rpc_get_int(json_rpc, "res_k", &res_value);
        if (ret == 0)
                printf("res_k=%d (should be the same as
previously)\n", res_value);
        else
                printf("No result? error %d\n", ret);

        abus_request_method_cleanup(abus, json_rpc);
        abus_cleanup(abus);

        return EXIT_SUCCESS;
}
```

5.3 Exemple d'export de données sur le bus

Ce troisième exemple détaille comment exporter des données depuis un service de façon à pouvoir être notifié coté client à chaque mise à jour du contenu de la variable :

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>

#include "abus.h"

int main(int argc, char **argv)
{
       abus_t *abus;
```

```
        const char *servicename = "exampleattrsvc";
        int i, ret;

        int my_int = 42;
        int my_auto_count = 0;

        abus = abus_init(NULL);

        ret = abus_decl_attr_int(abus, servicename, "tree.some_int", &my_int,
                                 ABUS_RPC_FLAG_NONE,
                                 "Some integer, for demo purpose");
        if (ret != 0) {
            abus_cleanup(abus);
            return EXIT_FAILURE;
        }

        /* No pointer is given for some_other_int, hence it will
           be auto-allocated by libabus, and initialized to zero.
           Access must be done through get and set.
         */
        abus_decl_attr_int(abus, servicename, "tree.some_other_int", NULL,
                                 ABUS_RPC_FLAG_NONE,
                                 "Some other integer, still for demo
purpose");

        abus_decl_attr_int(abus, servicename, "tree.auto_count", &my_auto_count,
                                 ABUS_RPC_FLAG_NONE,
                                 "Counter incremented every 5 seconds");

        /* do other stuff */
        for (i = 0; i < 1000; i++) {
                sleep(5);

                my_auto_count++;

                /* trigger event notification */
                abus_attr_changed(abus, servicename, "tree.auto_count");
        }

        abus_cleanup(abus);

        return EXIT_SUCCESS;
}
```

Code source du client associé :

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include "abus.h"

#define RPC_TIMEOUT 1000 /* ms */

int main(int argc, char **argv)
{
        abus_t *abus;
        int ret;
        const char *service_name = "exampleattrsvc";
        const char *attr_name;
        int my_int;

        if (argc < 2) {
                printf("usage: %s ATTR newintegervalue\n", argv[0]);
                printf("usage: ATTR: some_int|some_other_int\n");
                exit(EXIT_FAILURE);
        }

        abus = abus_init(NULL);

        /* attr name is taken from command line */
        attr_name = argv[1];

        ret = abus_attr_get_int(abus, service_name, attr_name, &my_
int, RPC_TIMEOUT);
        if (ret) {
                printf("RPC failed with error %d\n", ret);
                abus_cleanup(abus);
                exit(EXIT_FAILURE);
        }

        printf("Previous value: %s=%d\n", attr_name, my_int);

        my_int = atoi(argv[2]);

        ret = abus_attr_set_int(abus, service_name, attr_name, my_
int, RPC_TIMEOUT);
        if (ret) {
                printf("RPC failed with error %d\n", ret);
                abus_cleanup(abus);
                exit(EXIT_FAILURE);
        }
        printf("New value: %s=%d\n", attr_name, my_int);

        abus_cleanup(abus);

        return EXIT_SUCCESS;
}
```

Pour tester avec **abus-send** ce dernier exemple :

```
$ abus-send exampleattrsvc.get tree.some_int
$ abus-send exampleattrsvc.get tree.
$ abus-send exampleattrsvc.set tree.some_int:i=128
$ abus-send exampleattrsvc.get ""
```

Sinon, pour utiliser les événements émis à chaque modification :

```
$ abus-send exampleattrsvc.subscribe attr_changed%tree.auto_count
```

Ces trois exemples sont un petit aperçu du potentiel d'Abus, mais il est possible d'aller plus loin. Pour plus d'information, il est conseillé de consulter les exemples officiels disponibles sur *googlecode* : http://abus.googlecode.com/svn/doc/HEAD/html/examples.html.

6 Utilisation distante via un portail web

Abus ayant été développé en sockets locales **AF_UNIX** et non internet **AF_INET**, son utilisation s'est voulu locale et non distante.

Cependant, un mécanisme d'invocation distante reste possible via un serveur web à savoir via Lighttpd que nous devons installer au préalable :

```
sudo apt-get install lighttpd
```

Ensuite dans les exemples des sources d'Abus, commençons par éditer le fichier de configuration de Lighttpd fourni et adaptons le à notre chemin courant :

```
$ cd examples/
$ pwd
/home/tgayet/workspace/abus-read-only/examples
```

Le fichier de configuration **lighttpd.conf** doit être adapté de la façon suivante :

```
server.document-root = "/home/tgayet/workspace/abus-read-only/examples"
server.port                 = 8880
server.modules = ("mod_cgi")
server.modules += ( "mod_fastcgi" )
index-file.names = ( "index.html" )

mimetype.assign = (
".html"  => "text/html",
".css"  => "text/css",
".js"   => "text/javascript",
".gif"  => "image/gif",
".png"  => "image/png",
".jpeg"  => "image/jpeg",
)

cgi.assign = (
".sh" => "/bin/sh",
)

# /usr/share/doc/lighttpd-doc/fastcgi.txt.gz
# http://redmine.lighttpd.net/projects/lighttpd/wiki/
Docs:ConfigurationOptions#mod_fastcgi-fastcgi

## Start an FastCGI server for A-bus (needs the abus-cgi program)
fastcgi.server += ( "/rpc/" =>
  ((
    "bin-path" => "/home/tgayet/workspace/abus-read-only/tools/abus-cgi",
    "socket" => "/tmp/abus.fastcgi",
    "check-local" => "disable",
    "min-procs" => 1,
    "max-procs" => 5,
  ))
)
```

server.document-root définit le répertoire où sont localisés les pages HTML. **bin-path** définit l'emplacement du plugin **fastcgi** pour Lighttpd.

Démarrons le serveur web Lighttpd :

```
$ /usr/sbin/lighttpd -f ./lighttpd.conf
2013-10-09 15:38:25: (log.c.166) server started
```

Vérifions le lancement :

```
$ ps aux | grep -i lighttpd
tgayet     909  0.0  0.0   4920   724 ?        S    15:38   0:00 /usr/sbin/
lighttpd -f ./lighttpd.conf
$ sudo apt-get install sockstat
$ sockstat -P 909
USER     PROCESS      PID      PROTO  SOURCE ADDRESS  FOREIGN ADDRESS  STATE
tgayet   lighttpd      909      tcp4   *:8880           *:*             LISTEN
```

Le serveur web est donc bien lancé, la page qui s'affichera par défaut sera **index.html**.

Pour le test, le service **example-service** doit toujours être lancé en tâche de fond. Ce dernier recevra les sollicitations du plugin **fcgi** en provenance des pages web HTML5.

Lançons un navigateur web depuis l'adresse IP externe sur le port 8880. En local on peut utiliser l'adrese suivante : **http://localhost:8880**.

Les fichiers important sont les suivant :

```
$ ls -al example/
(...)
-rw-rw-r--  1 tgayet tgayet  1860 oct.  9 10:38 index.html
-rw-rw-r--  1 tgayet tgayet 93868 oct.  9 10:38 jquery.js
-rw-rw-r--  1 tgayet tgayet  9551 oct.  9 10:38 jquery.jsonrpc.js
-rw-rw-r--  1 tgayet tgayet   837 oct.  9 15:38 lighttpd.conf
```

La page **index.html** doit afficher le contenu suivant :

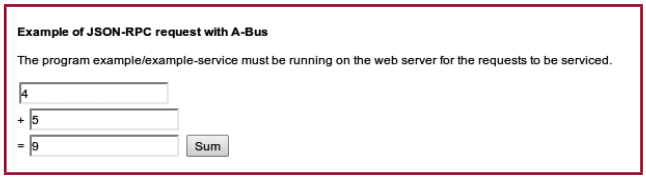


Fig. 9

Pendant l'exécution, le service **examplesvc** affichage sur **STDOUT** les traces d'exécutions des différents appels :

```
$ ## 22319 <- 00175:73 {"jsonrpc":"2.0","method":"examplesvc.sum","id":1,
"params":{"a":4,"b":5}}
## svc_sum_cb: arg=sumator cookie, ret=0, a=4, b=5, => result=9
## 22319 -> 00175:49 {"jsonrpc":"2.0","result":{"res_value":9},"id":1}
```

En saisissant 4 pour **a**, 5 pour **b** et en pushant les données via un plugin **json-rpc** au serveur web, les données seront redirigées vers le plugin **fcgi** abus qui sert de proxy avant de lancer l'appel sur le bus puis de renvoyer la réponse au client web. Le plugin **fcgi** de lighttpd se comporte comme la commande **abus-send** c'est a dire qu'il communique en direct avec le bus.

Le contenu de la page web est la suivante :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.
w3.org/TR/html4/loose.dtd">
<html>
 <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <style type="text/css">
    <!--
    * { font-family: Helvetica, Verdana, Arial ; font-size:12px ;
color:#000000 }
    -->
  </style>
    <title>A-Bus JSON-RPC example</title>
    <!-- <link href="layout.css" rel="stylesheet" type="text/css"> -->
    <script language="javascript" type="text/javascript" src="jquery.js"></
script>
```

```
    <script language="javascript" type="text/javascript" src="jquery.
jsonrpc.js"></script>
 </head>
    <body>
  <h2>Example of JSON-RPC request with A-Bus</h2>
  <p>
    The program example/example-service must be running on the web server
for the requests to be serviced.
  </p>
  <div>
    <form>
      <input type=text id="firstBox" value=""><br>
      + <input type=text id="secondBox" value=""><br>
      = <input type=text id="thirdBox">
      <input id="sumUpdate" type="button" value="Sum">
    </form>
  </div>

  <script type="text/javascript">
  <!--
    $(function ()
    {
      $.jsonRPC.setup({
          endPoint: '/rpc/fastrpc.sh',
        namespace: 'examplesvc'
      });

      $("#sumUpdate").click(function ()
      {
        // Récupération des deux valeurs dans le formulaire
        var vala = parseInt($("#firstBox").val());
        var valb = parseInt($("#secondBox").val());

        // Invocation de la RPC sum du service examplesvc
        $.jsonRPC.request('sum',
        {
          params: {"a":vala, "b":valb},
          success: function(result)
          {
            if (result.result == null || result.result.res_value == null)
              alert("Invalid result from JSON-RPC: "+result);
            else
              $("#thirdBox").val(result.result.res_value);
          },
          error: function(result)
          {
            var msg = '';
            if (result.error.message != null)
              msg += result.error.message;
            else
              msg += result.error;
            alert('Error: unable to get sum: '+msg);
          }
        });
      });

    });
  //-->
  </script>
 </body>
</html>
```

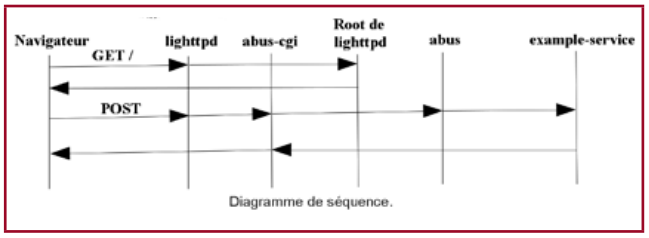


Fig. 10 : Diagramme de séquence

Le **endpoint** servira de préfixe à Lighttpd pour rediriger les requêtes vers le plugin **fcgi** abus. Donc toute requête qui commencera par **rpc** sera donc rediriger et interprété vers abus. Tout ce qui suit comme **fastrpc.sh** ne sert pas.

param permet de regrouper les paramètres à envoyer au service et **result.result.res_value** sert à récupérer la valeur de retour (Fig. 10).

Conclusion

Bon voilà, un petit diaporama sur Abus qui je l'espère vous aura donné envie de l'utiliser. C'est un premier snapshot et la roadmap est bien fournie. ■

Liens

- http://code.google.com/p/abus/
- http://code.google.com/p/googletest/
- http://zeromq.org/
- http://code.google.com/p/rpcz/
- http://code.google.com/p/protobuf/
- http://msgpack.org/
- https://wiki.gnome.org/JsonGlib
- https://developers.google.com/protocol-buffers/docs/overview?hl=fr&csw=1
- http://fr.wikipedia.org/wiki/Synchronisation_(multit%C3%A2ches)
- http://proton.inrialpes.fr/~krakowia/Enseignement/M2P-GI/Flips/4-MessagesEvents-4pp.pdf
- http://pwet.fr/man/linux/administration_systeme/ipcs
- http://pwet.fr/man/linux/administration_systeme/ipcrm
- http://www.angryredplanet.com/~hackbod/openbinder/
- https://lkml.org/lkml/2009/6/25/31
- http://events.linuxfoundation.org/images/stories/slides/abs2013_gargentas.pdf
- http://fr.slideshare.net/jserv/android-ipc-mechanism

RÉUTILISER DU CODE C/C++ NATIF SOUS ANDROID AVEC LE NATIVE DEVELOPMENT KIT (NDK)

par Sylvain Saurel [Ingénieur d'études Java / Java EE]

Apparu dès les débuts de la plateforme, le Native Development Kit (NDK) offre la possibilité aux développeurs d'embarquer au sein de leurs applications Android des bibliothèques C ou C++ ayant déjà fait leurs preuves. Bien entendu, l'utilisation du NDK reste réservée à des applications bien spécifiques pour lesquelles le recours au NDK apporte des avantages indéniables. Au cours de cet article, nous allons voir comment mettre en oeuvre le NDK en réalisant une application s'appuyant sur une bibliothèque C de traitement d'images.

Le NDK regroupe l'ensemble des outils proposés par Google pour réutiliser des parties de codes natifs C/C++ au sein d'applications Android. Compte tenu du nombre important de bibliothèques C/C++ existantes et reconnues dans divers domaines, cette possibilité peut s'avérer très intéressante pour les développeurs de certains types d'application. En effet, la première chose à prendre en compte avec le NDK est qu'il ne sera pas forcément bénéfique à la plupart des applications. Il ne faudra, par exemple, pas recourir au NDK pour la simple raison que l'on préfère développer en C ou C++ plutôt qu'en Java. En revanche, utilisé à bon escient dans des applications effectuant des traitements d'images ou du signal, des simulations physiques ou bien pour les moteurs physiques de certains jeux, le NDK se révèle pertinent. Ainsi, il est destiné en priorité à des traitements utilisant de manière intensive le CPU et ne nécessitant que peu d'allocations mémoires. Dans tous les cas, il reviendra au développeur de mesurer si les avantages prennent le pas sur les inconvénients induits par la réutilisation de bibliothèques C/C++ et notamment l'augmentation de la complexité du code d'une application. Comme toujours en informatique, le pragmatisme est de mise quelle que soit la technologie considérée.

1 Fonctionnement du NDK

Au coeur de l'architecture du NDK, on retrouve la technologie JNI (Java Native Interface) bien connue des développeurs Java qui est également intégrée au sein du SDK Android. JNI permet de faire le lien entre le code natif C/C++ et le code Java d'une application. Ainsi, lors de l'exécution d'une application Android ayant recours au NDK, lorsque la machine virtuelle Dalvik appelle une fonction JNI, elle lui passe en paramètre 2 pointeurs Java, le premier sur un objet de type JNIEnv et le second sur un objet quelconque. Le pointeur de type JNIEnv est une structure proposant une interface vers Dalvik incluant toutes les fonctions permettant d'interagir avec la machine virtuelle et les objets Java. Une fonction JNI aura ainsi l'allure suivante :

```
JNIEXPORT void JNICALL Java_ClassName_
MethodName(JNIEnv *env, jobject obj) {
 // ...
}
```

Globalement, une application Android NDK est ainsi similaire à une application classique à cela près qu'elle s'appuie sur JNI et embarque les bibliothèques C/C++ utilisées (figure 1).

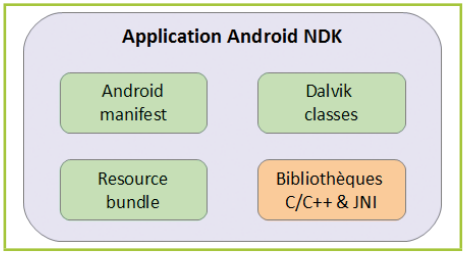


Fig.1 : Composants d'une application NDK

2 Réalisation d'une application NDK

Afin de mettre en œuvre le NDK, nous allons réaliser une application Android pour laquelle le recours à du code natif procure un avantage indéniable. Comme expliqué précédemment, plusieurs possibilités existent : utilisation de codes natifs pour un jeu vidéo, traitement du signal ou encore traitement d'images. Ces différents cas d'utilisation se prêtent particulièrement bien au NDK puisque nécessitant de nombreux calculs provoquant une utilisation intensive du CPU mais n'allouant que peu d'espace mémoire. Notre application va donc proposer d'afficher une image PNG et d'y appliquer différents types de traitements d'images tels qu'une conversion en niveaux de gris ou une détection des contours. Les bibliothèques C/C++ de manipulation d'images sont nombreuses de même que les exemples d'algorithmes pour ce type d'opérations. Il va donc s'avérer aisé d'écrire du code implémentant ces traitements.

3 Mise en place de l'environnement

Le premier pré-requis est bien entendu de posséder une installation récente du SDK Android sur son poste de travail. Par récente, nous entendons supérieure à Android 4.0 soit le niveau 14 du SDK. Ce dernier est téléchargeable à l'adresse suivante : http://developer.android.com/sdk/. Il faut ensuite procéder à l'installation du NDK disponible ici : http://developer.android.com/tools/sdk/ndk/. Par la suite, nous définirons par **<ndk>** le chemin d'installation du NDK. Enfin, au niveau de l'IDE nous nous appuierons sur Eclipse et le plugin ADT fourni par Google. L'installation de ce plugin se fait via l'update site Eclipse suivant : https://dl-ssl.google.com/android/eclipse/.

4 Création de l'environnement

Sous Eclipse, la création de l'application se fait via la création d'un nouveau projet d'Application Android dont la version minimum du SDK doit être supérieure à Android 4.0. La structure de l'application ainsi créée est somme toute classique (figure 2).

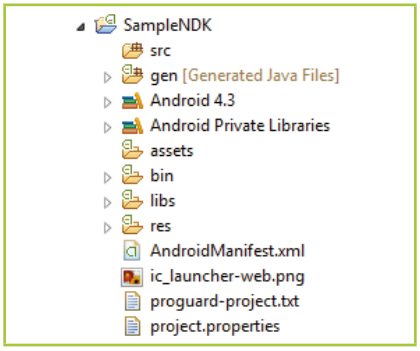


Fig.2 : Structure du projet NDK

L'approche retenue pour l'application est de réaliser en premier la partie Android classique avec la création d'une activité et la mise en place de l'interface graphique avant par la suite de réaliser le pont avec la bibliothèque C / C++ de traitement d'images via JNI. Le but étant également de rendre transparente au sein d'Eclipse l'utilisation du NDK durant le développement.

5 Interface graphique

L'application se compose d'une activité principale s'affichant à son lancement. Cette activité s'appuie sur un layout spécifique possédant une liste de boutons implémentant chacun un des traitements d'images proposés ainsi que d'une image sur laquelle les traitements seront appliqués. Sur chaque bouton, on définit la méthode **handler** qui sera appelée du côté de l'activité lors d'un évènement click. L'image étant ensuite mise à jour après l'application d'un traitement. Le contenu de ce layout est le suivant :

```
<LinearLayout xmlns:android="http://schemas.
android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:orientation="vertical" >

 <LinearLayout
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:gravity="center"
  android:orientation="horizontal" >

  <Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick="onRaz"
   android:text="@string/raz" />

  <Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick="onConvertToGray"
   android:text="@string/convertGray" />

  <Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick="onFindContours"
   android:text="@string/findContours" />

  <Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick="onBrightMinus"
   android:text="@string/brightMinus" />

  <Button
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:onClick="onBrightPlus"
   android:text="@string/brightPlus" />
 </LinearLayout>

 <ImageView
  android:id="@+id/image"
 android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center_
vertical|center_horizontal"
  android:scaleType="centerCrop" />
</LinearLayout>
```

6 Activité principale

Nommée sans surprise **MainActivity**, l'activité principale charge le layout XML présenté précédemment et définit les méthodes listeners d'évènements click sur les différents boutons. Ces derniers servant à associer les traitements d'images correspondants à l'interface

utilisateur. Outre ces lignes de code classiques, l'activité permet de charger la bibliothèque native que nous allons utiliser via l'appel suivant :

```
static {
  System.loadLibrary("ProcImage");
}
```

La définition des méthodes natives se fait de la même manière que dans un programme Java desktop via l'utilisation du mot clé native :

```
public native void convertToGray(Bitmap bitmapIn,Bitmap bitmapOut);
```

Le code de l'activité ainsi définie est donc le suivant :

```
public class MainActivity extends Activity {
 private Bitmap bmpOrigin;
 private Bitmap bmpGray;
 private Bitmap bmpResult;
 private ImageView image;

 static {
  System.loadLibrary("libProcImage");
 }

 public native void convertToGray(Bitmap in, Bitmap out);
 public native void changeBrightness(int dir, Bitmap bmp);
 public native void findContours(Bitmap in, Bitmap out);

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  image = (ImageView) findViewById(R.id.image);
  // chargement bitmap
  BitmapFactory.Options options = new BitmapFactory.Options();
  // bitmap stocké en 24 bits
  options.inPreferredConfig = Config.ARGB_8888;
  bmpOrigin = BitmapFactory.decodeResource(getResources(),
R.drawable.sampleimage, options);

  if (bmpOrigin != null) {
   image.setImageBitmap(bmpOrigin);
  }
 }

 public void onRaz(View v) {
  image.setImageBitmap(bmpOrigin);
}

 public void onFindContours(View v) {
  bmpGray = Bitmap.createBitmap(bmpOrigin.getWidth(), bmpOrigin.
getHeight(), Config.ALPHA_8);
  bmpResult = Bitmap.createBitmap(bmpOrigin.getWidth(), bmpOrigin.
getHeight(), Config.ALPHA_8);
  // niveaux de gris
  convertToGray(bmpOrigin, bmpGray);
  // contours
  findContours(bmpGray, bmpResult);
  image.setImageBitmap(bmpResult);
 }

 public void onConvertToGray(View v) {
  bmpResult = Bitmap.createBitmap(bmpOrigin.getWidth(), bmpOrigin.
getHeight(), Config.ALPHA_8);
  convertToGray(bmpOrigin, bmpResult);
  image.setImageBitmap(bmpResult);
 }

 public void onBrightPlus(View v) {
  if (bmpResult != null) {
   changeBrightness(1, bmpResult);
   image.setImageBitmap(bmpResult);
  }
 }

 public void onBrightMinus(View v) {
   if (bmpResult != null) {
   changeBrightness(2, bmpResult);
   image.setImageBitmap(bmpResult);
  }
 }
}
```

Pas de difficulté particulière dans ce code si ce n'est qu'il faut bien prendre garde à charger l'image PNG sur laquelle nous allons effectuer les traitements comme un bitmap avec couleurs stockées en 24 bits à l'aide de l'option suivante :

```
options.inPreferredConfig = Config.ARGB_8888;
```

Ce chargement du bitmap en 24 bits est essentiel puisque la bibliothèque de traitement d'images sur laquelle nous nous appuierons par la suite travaille sur des bitmaps de ce type. Le reste de l'activité consistant simplement à gérer les appels aux fonctions natives de traitement d'images lors du click sur l'un des boutons de l'IHM de l'application.

7 Bibliothèque native

La bibliothèque native que nous allons créer se base sur le fichier source **procimage.c**. Ce fichier doit être placé au sein d'un répertoire **jni** à la racine du projet. Le NDK installé précédemment fournit différents fichiers headers qu'il faudra inclure au sein du fichier **procimage.c** afin d'accéder aux fonctionnalités liées à JNI mais également à la manipulation de bitmaps en NDK. La présentation des traitements d'images utilisés par la suite n'étant pas l'objet de cet article, nous nous contentons de détailler simplement une seule des fonctions JNI définie :

```
#include <jni.h>
#include <android/bitmap.h>

typedef struct {
 uint8_t alpha;
 uint8_t red;
 uint8_t green;
 uint8_t blue;
```

```
} argb;

JNIEXPORT void JNICALL Java_com_ssaurel_samplendk_MainActivity_
convertToGray(JNIEnv * env, jobject obj, jobject bitmapcolor,jobject
bitmapgray) {
 AndroidBitmapInfo infocolor;
 void* pixelscolor;
 AndroidBitmapInfo infogray;
 void* pixelsgray;
 int ret;
 int y;
 int x;

 AndroidBitmap_getInfo(env, bitmapcolor, &infocolor);
 AndroidBitmap_getInfo(env, bitmapgray, &infogray);
 AndroidBitmap_lockPixels(env, bitmapcolor, &pixelscolor);
 AndroidBitmap_lockPixels(env, bitmapgray, &pixelsgray));

 for (y = 0; y < infocolor.height; y++) {
  argb * line = (argb *) pixelscolor;
  uint8_t * grayline = (uint8_t *) pixelsgray;
  for (x = 0; x < infocolor.width; x++) {
   grayline[x] = 0.3 * line[x].red + 0.59 * line[x].green +
0.11*line[x].blue;
  }

  pixelscolor = (char *)pixelscolor + infocolor.stride;
  pixelsgray = (char *) pixelsgray + infogray.stride;
 }

 AndroidBitmap_unlockPixels(env, bitmapcolor);
 AndroidBitmap_unlockPixels(env, bitmapgray);
}
```

Définie au début du fichier source, la structure **argb** sert à représenter les composantes couleurs d'un pixel au format 32 bits. Dans la définition de la fonction JNI **convertToGray**, on reconnaît la signature habituelle d'une fonction JNI avec notamment la présence des 2 pointeurs comme premiers paramètres qui sont rajoutés par la machine virtuelle Dalvik lors de l'appel depuis le code Java. En outre, le nom de la fonction respecte la convention JNI avec Java comme préfixe suivi du nom qualifié de la classe avec laquelle le lien doit être fait pour la fonction. Enfin, on notera que les fonctions préfixées par **AndroidBitmap_** proviennent de la bibliothèque des fonctions natives fournies par le NDK Android pour la manipulation de bitmaps.

8 Compilation

Afin de produire la bibliothèque de traitement **ProcImage** et le code JNI spécifique qui sera utilisé par l'application Android, il est nécessaire de compiler le fichier source **procimage.c**. Pour ce faire, le NDK impose la définition d'un fichier makefile spécifiant comment les fichiers sources devront être compilés. Le fichier makefile se nomme **Android.mk** et est à placer dans le répertoire **jni**. Dans le cadre de notre projet, son contenu est le suivant :

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := ProcImage
LOCAL_SRC_FILES := procimage.c
LOCAL_LDLIBS    := -ljnigraphics
include $(BUILD_SHARED_LIBRARY)
```

Ce makefile définit tout d'abord le répertoire de travail avant de vider les variables utilisées. La variable **LOCAL_MODULE** permet de spécifier le nom de la bibliothèque qui sera **ProcImage** dans notre cas. **LOCAL_SRC_FILES** définit les fichiers sources à prendre en entrée ce qui se limite au seul fichier **procimage.c** ici. De plus, **LOCAL_LDLIBS** est nécessaire puisque nous souhaitons préciser au compilateur que nous utilisons la bibliothèque **libjnigraphics.so**. Enfin, la dernière ligne précise le type de fichier en sortie. Ici, il s'agit d'une bibliothèque partagée.

Un autre makefile plus global peut être défini bien qu'il reste optionnel. Le makefile **Application.mk** sert à définir ou surcharger des variables liées au NDK qui seront appliquées durant la compilation. Ici, nous définissons la version de la plateforme utilisée mais également le fait que la bibliothèque partagée aura une version à destination des processeurs ARM en général et une version optimisée pour les processeurs ARM V7 comme suit :

```
APP_PLATFORM := android-18
APP_ABI := armeabi armeabi-v7a
```

Les bibliothèques compilées seront de fait présentes au sein du répertoire **libs** du projet dans les répertoires cibles **armeabi** et **armeabi-v7a**. La compilation de la bibliothèque partagée se fait en exécutant la commande **ndk-build** présente à la racine du répertoire **<ndk>**.

9 Intégration dans Eclipse

Recourir à un terminal pour lancer la commande **ndk-build** n'est pas forcément la chose la plus pratique en termes de productivité. Il est donc intéressant d'intégrer le build par le NDK directement au sein d'Eclipse. Ceci dans le but de gagner du temps et d'éviter d'éventuelles erreurs de rafraîchissement au sein de l'IDE. En effet, sans cette intégration, il serait nécessaire d'écrire le code C/C++ avant de quitter l'IDE pour lancer la compilation puis de revenir dans l'IDE sans oublier d'effectuer une mise à jour des fichiers du projet. Cette dernière étape ayant de grandes chances d'être oubliée par le développeur conduisant à des erreurs et à une perte de temps certaine. Au sein d'Eclipse, nous allons donc intégrer la compilation NDK via l'utilisation d'un builder spécifique. Pour ce faire, il faut aller dans les propriétés du projet courant SampleNDK puis cliquer sur l'entrée Builders

du menu principal. Parmi les différents builders déjà définis (figure 3), il faut rajouter en premier un nouveau builder de type Program. Ce builder utilise la commande **ndk-build** présente à la racine du répertoire **<ndk>** et travaille dans le répertoire jni à la racine du projet (figure 4). Dans l'onglet Build Options, il reste à préciser que le builder est à lancer durant les builds automatiques.

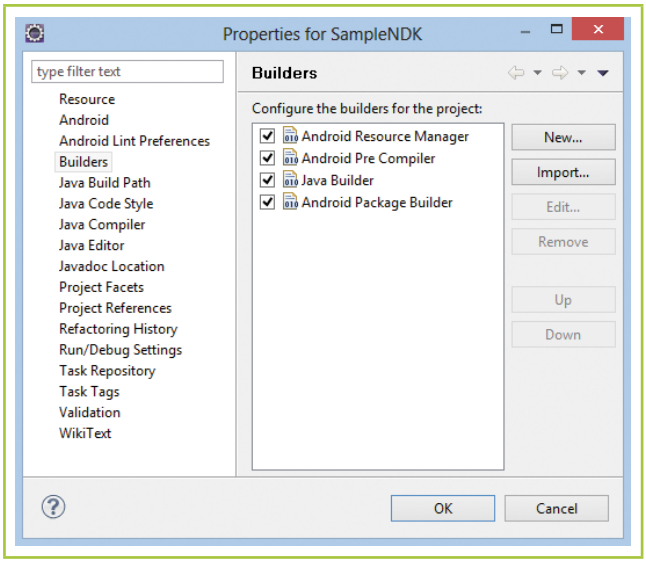


Fig.3 : Builders standards d'un projet Android

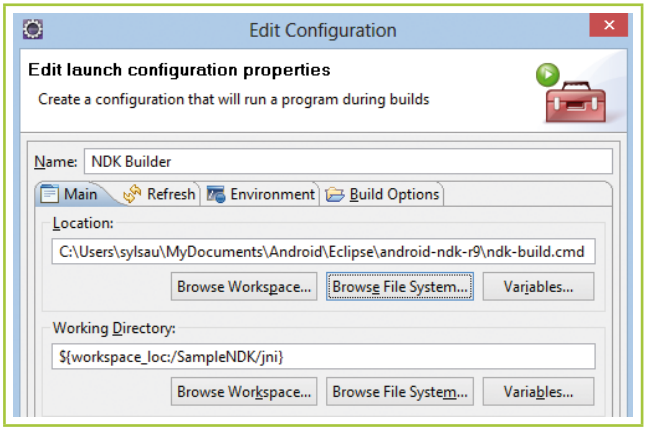


Fig.4 : NDK Builder pour le projet

Une fois les modifications du builder appliquées, le build automatique d'Eclipse permet son exécution et on peut visualiser le résultat dans la console Eclipse (figure 5).

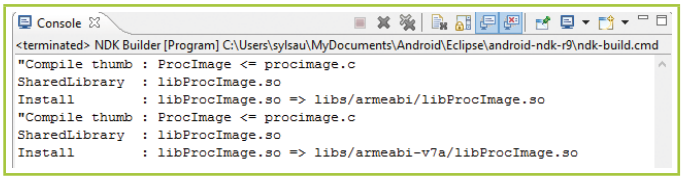


Fig.5 : Création de la bibliothèque via le NDK Builder

10 Exécution de l'application

La compilation réalisée, il ne reste plus qu'à exécuter l'application afin de tester son bon fonctionnement. L'exécution de l'application se réalise simplement en faisant un Run As Android Application sous Eclipse. Les figures 6 et 7 permettant de voir une image affichée ainsi que l'image correspondante en niveaux de gris. Il est bon de noter qu'avec l'intégration de la commande **ndk-build** en tant que Builder du projet, l'export de l'APK de l'application se fera également de manière classique comme pour une application Android standard sans travail supplémentaire.

Fig.6 : Image de départ

Fig.7 : Image en niveaux de gris

Conclusion

Reposant sur la technologie JNI développée à l'origine par Sun, le NDK offre aux développeurs Android la possibilité de réutiliser aisément du code natif au sein d'applications. L'intégration et l'utilisation de ce code se réalise aisément comme nous avons pu le voir au cours de cet article. Néanmoins, cette relative facilité ne doit pas faire perdre de vue la complexité induite par ce mode de fonctionnement. En outre, les performances peuvent se révéler désastreuses lorsque le NDK est utilisé dans des cas d'utilisation inappropriés. Ainsi, son emploi doit se limiter à des traitements faisant un usage intensif du CPU et à la réutilisation de bibliothèques C/C++ ayant déjà fait leurs preuves. Pour ces cas d'utilisation précis, le NDK apporte des avantages indéniables supplantant les inconvénients de l'approche bien que là encore, les améliorations constantes apportées à la machine virtuelle Dalvik laissent à penser que si une API du framework existe pour ces traitements spécifiques, il sera avantageux de l'utiliser en lieu et place du NDK. ■

ET SI VOUS PASSIEZ À LA PUISSANCE CLOUD ?

VAR

Distinguez-vous de vos concurrents en valorisant votre offre.

SSII

Profitez d'infrastructures IaaS déployées en un clin d'œil.

ÉDITEURS

Passez au SaaS en vous appuyant sur notre savoir-faire.

PROGRAMME PARTENAIRE ARUBA CLOUD

Contactez-nous

- 2 niveaux de marque blanche disponibles.
- Gestion simple, souple et performante de l'infrastructure (publique, privée ou hybride).
- Modélisation et activation immédiate de votre datacenter virtuel dans le pays de votre choix.
- Interface client totalement personnalisable.
- Facturation en " Pay as you go ".
- Finesse dans la gestion des droits utilisateurs.

Aruba, le bon partenaire pour bénéficier de la puissance d'un acteur majeur qui considère que chaque client, dans chaque pays, est unique. **MY COUNTRY. MY CLOUD.**

aruba CLOUD

arubacloud.fr | TÉL : 0810 710 300
(COÛT D'UN APPEL LOCAL)

À vous de choisir !

LINAGORA

est fière de faire du vrai logiciel libre
et vous souhaite une bonne année 2014 !

www.linagora.com