PROGRAMMEZ.COM

PROGRAMMEZ!

Mensuel n°181 - Janvier 2015

le magazine du développeur

Je débute avec Xamarin

Des apps natives pour Android, iOS, Windows Phone avec un seul code !

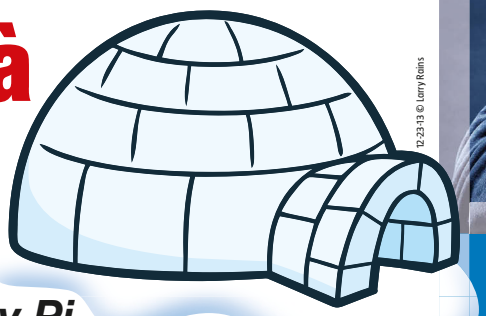
Hacker : niveau demi-dieu

L'essentiel de la conférence No Such Con '14

Les conséquences de l'informatique quantique

Protéger son code Android

Coding à la neige



Démarrer un projet Raspberry Pi ou Arduino

Facile comme un capteur de température !

Matériel

Monter son PC 4K

Scala

un langage objet et fonctionnel au coeur de Java

Programmation avancée

Pensez multithread

OpenMP, MPI : savoir les utiliser

Traquer les bugs dans un code parallèle

Comment optimiser son code avec Java 8

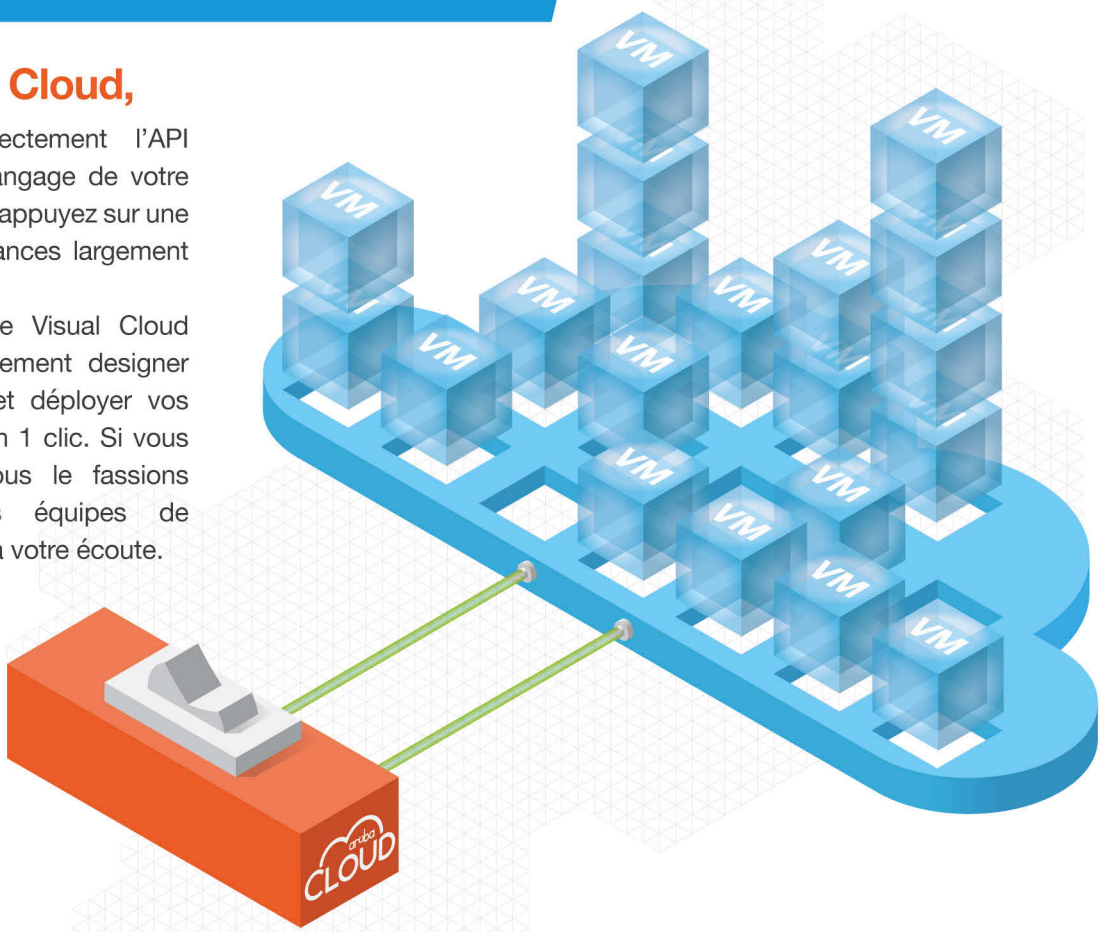
M 04319 - 181S - F: 5,95 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 6,45 € SUISSE 12 FS - LUXEMBOURG 6,45 € - DOM Surf 6,90 € Canada 8,95 $ CAN - TOM 940 XPF - MAROC 50 DH

Je veux créer **20 cloud serveurs** *en un clic* **Comment faire ?**

Avec Aruba Cloud,

vous utilisez directement l'API Aruba depuis le langage de votre choix et vous vous appuyez sur une base de connaissances largement documentée.
Grâce à l'interface Visual Cloud vous pouvez également designer votre datacenter et déployer vos serveurs virtuels en 1 clic. Si vous souhaitez que nous le fassions pour vous, nos équipes de Consultance sont à votre écoute.



- 3 hyperviseurs
- 6 datacenters en Europe
- APIs et connecteurs
- 70+ templates
- Contrôle des coûts

*MON PAYS, MON CLOUD.

1 Quitte à choisir une infrastructure IaaS, autant prendre la plus performante!
Aruba Cloud est de nouveau **N°1 du classement des Cloud**
JDN / CloudScreener / Cedexis (octobre 2014)

Contactez-nous! 0810 710 300 www.arubacloud.fr

aruba CLOUD

Cloud Public | Cloud Privé | Cloud Hybride | Cloud Storage | Infogérance

MY COUNTRY. MY CLOUD.*

Le boss.
ftonic@programmez.com

« Je suis ton père »

Quelques jours avant les fêtes de fin d'année, nous avons perdu un des personnages les plus importants du monde numérique : Ralph Baer. Souvenez-vous. En 1971/2, la première machine de jeux sort : Brown Box (ou Magnavox Odyssey). Les premiers prototypes remontent à 1968 – 69. Le jeu Pong lui doit beaucoup. Et Atari va connaître un énorme succès dès 1973. Ralph est le créateur du célèbre jeu « Simon ».

Regardez cette vidéo de Ralph avec sa « console : http://www.youtube.com/watch?v=scaLx1l_j5w#t=39

Il avait 92 ans. Tous les joueurs, nerds et geeks lui doivent beaucoup. Respect numérique !

Presque au même moment, Sony et les gamers fêtaient un événement historique qui a révolutionné la console et le jeu vidéo : les 20 ans de la PlayStation. Le premier modèle était sorti en décembre 1994. Console culte, la PlayStation a permis des avancées techniques énormes sur les processeurs et les technologies 3D. Et même si aujourd'hui, la PS souffre avec la XBOX, la console Sony reste une référence.

Tous ces événements donnent une saveur un peu spéciale à la rubrique matérielle de ce numéro. Vous allez découvrir comment un hardcore gamer a monté et créé un PC de gaming qui explose tout : un PC 4K ! Ames sensibles, passez votre chemin! Une config hallucinante à... 4 000 € !

Dans ce dernier numéro de l'année, et pour bien débuter 2015, nous avons concocté un réveillon équilibré : Arduino, Raspberry Pi, développement HPC et multi-core, Scala, Xamarin, Cordova, du hack quantique...

Cela fait un peu plus d'un an que Programmez ! a changé radicalement : nouveau patron, nouvelle maquette, nouveaux contenus, nouveau site Web, premières apps. Pour 2015, nous allons continuer à nous adapter à vos besoins, à vos demandes. Sans vous, Programmez ! n'existerait pas. Nous vous remercions de votre soutien et de vos messages et commentaires.

Je tiens à remercier tous les contributeurs à Programmez ! pour leurs articles, leurs conseils, leurs propositions. Merci à l'équipe Programmez ! qui travaille dur chaque jour.

N'HÉSITEZ PAS !

Vous voulez partager votre expérience ? Une critique (bonne ou mauvaise) ? Proposer des articles ? Toute l'équipe de Programmez ! est à votre écoute : ftonic@programmez.com, redaction@programmez.com

Rendez-vous en 2015 !

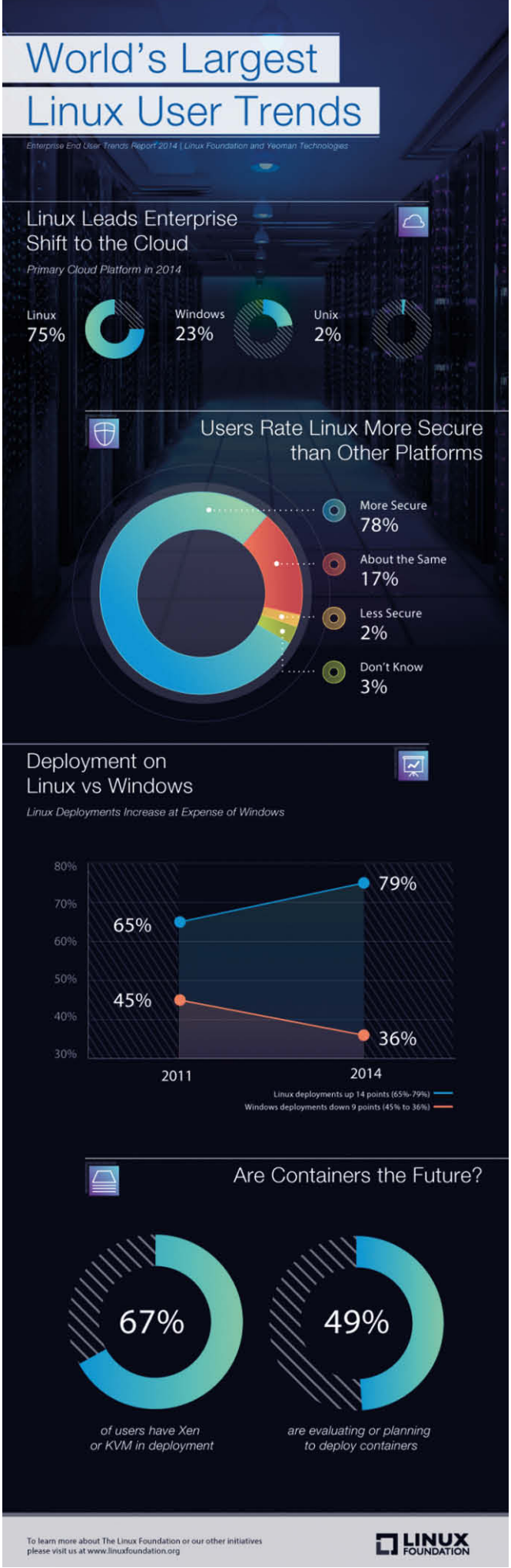
À LIRE DANS LE PROCHAIN NUMÉRO n°182 en kiosque le 31 Janvier 2015

Carrière
Développeur, le plus beau métier du monde !

3D
Découvrir et utiliser Unity3D

Question
Qu'est-ce qu'une Machine Learning ?

Très bonne année geek 2015



QUELS LANGAGES PAIENT LE PLUS AUX ETATS-UNIS ?

Le site qz.com propose un top 12 des langages payant le plus... en $, en moyenne, salaire annuel brut.

Ruby on Rails	109 460
Objective-C	108 225
Python	100 717
Java	94 908
C++	93 502
JavaScript	91 461
C	90 134
R	90 055
C#	89 074
VB	85 962
SQL	85 511
PERL	82 513

Ce panorama n'est qu'une image approximative du marché des développeurs américains. Le cas Ruby on Rails est intéressant même si ce n'est pas un langage à proprement parler. Les spécialistes de Rails sont assez peu nombreux, ceci peut expliquer le chiffre élevé. Pour Objective-C, cela prouve la popularité des plateformes Apple et qu'il y a un manque de compétences pour une demande relativement forte. Après il faut fortement nuancer selon les Etats et le type de société. Mais intrinsèquement, un bon développeur américain gagne plus qu'un développeur français.

Rovio perd une centaine d'oiseaux : licenciements prévus

Docker c'est bien mais **CoreOS**, pourtant fervent partisan de la startup, veut créer ses propres conteneurs...

Les serveurs chinois débarquent !

84 millions $: le salaire annoncé pour le nouveau boss de Microsoft

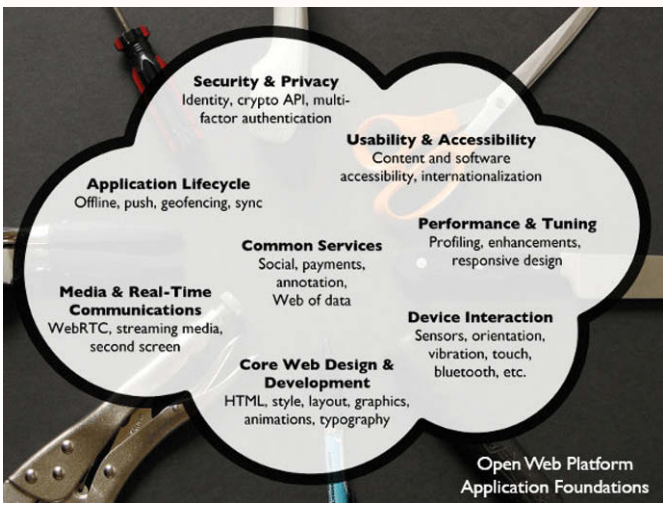
Samsung ne veut pas payer **Apple** : il est où le chèque de 900 millions $?

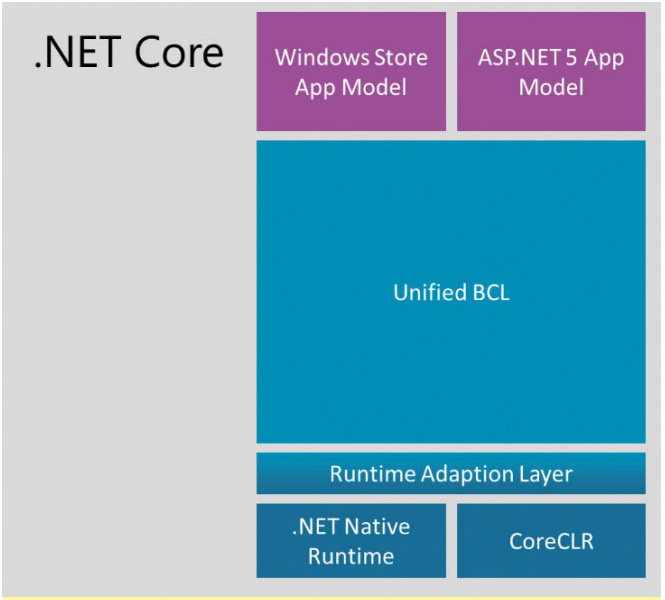
Après HTML 5, Open WebPlatform

Alors que HTML5 est officiellement une recommandation du W3C, le consortium entame un autre chantier : l'open web platform. Le but : proposer une plate-forme de développement et déploiement pour les applications, s'appuyant sur des composants et technologies standards et assurer une interopérabilité maximale. Pour Jeff Jaffe, c'est la prochaine priorité du W3C.

La plateforme s'articule autour de plusieurs piliers : sécurité, un cœur technique (web design / développement), interaction vers / entre terminaux, la gestion du cycle de vie des applications, media et communication temps réel, performances, services, accessibilité.

Présentation complète : http://www.w3.org/blog/2014/10/application-foundations-for-the-open-web-platform/



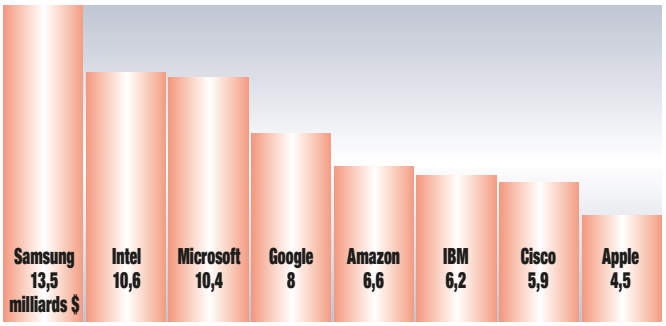


.NET CORE ?

Lis ce post : http://blogs.msdn.com/b/dotnet/archive/2014/12/04/introducing-net-core.aspx

Qui investit le plus en recherche et développement ?

La R&D est un des enjeux des entreprises, notamment dans le monde informatique. La R&D ne se voit pas forcément immédiatement et parfois, des projets entiers sont abandonnées. Ce n'est pas parce que son R&D est énorme que l'on est plus efficace...



Source : microsoft-news.com, compilation de chiffres.

Aurora :

Amazon veut concurrencer Oracle, MySQL, MariaDB, SQL Server avec son propre service de données !

Windows 10,

version grand public prévue pour janvier. (pré-version)

XAML OU HTML 5 :

à vous de choisir, http://codefoster.com/htmloverxaml/

RIP les captcha !

Google décide de supprimer les captcha. Faisons une minute de silence pour nos amis les robots.

Google

veut conquérir un nouveau marché : les enfants !

Angleterre

une taxe Google pour mieux taxer les multinationales du numérique.

Stuxnet

rappelez-vous de l'attaque informatique contre les sites nucléaires iraniens. Aujourd'hui, on soupçonne l'Iran de faire la même chose avec Cleaver...

Miroir, miroir, quel est le langage le plus utilisé ?

L'index TIOBE donne chaque mois les langages les plus utilisés par les développeurs. Il s'agit d'un indicateur basé sur les recherches web. Des % à manipuler avec précaution.

Nov 2014	Nov 2013	Tendance	Langage	%	Evolution en %
1	1	↓	C	17.469%	-0.69%
2	2	↓↓	Java	14.391%	-2.13%
3	3	↓	Objective-C	9.063%	-0.34%
4	4	↓↓	C++	6.098%	-2.27%
5	5	↓	C#	4.985%	-1.04%
6	6	↓↓	PHP	3.043%	-2.34%
7	8	↓	Python	2.589%	-0.52%
8	10	↑	JavaScript	2.088%	+0.04%
9	12	↑	Perl	2.073%	+0.55%
10	11	↑	Visual Basic .NET	2.061%	+0.09%

Le podium reste identique : C, Java et Objective-C. Tous les langages baissent exceptés JavaScript, Pert et VB.Net. Parmi les langages novateurs et nouveaux, notons le langage R qui pointe à 12e place, F# arrive 16e, Swift est 18e et Dart 20e.

agenda

janvier

La communauté Microsoft de Strasbourg organise une conférence le Mardi 6 Janvier 2015 à 18h dans les locaux de l'eXia.Cesi de Strasbourg.
Le contenu de la présentation sera :
- Azure Websites : comme son nom l'indique, cela concernera tout ce qui touche au déploiement et gestion d'un site web : de Visual Studio à TFS Online.
- Azure Mobile Service + Service Bus pour la gestion des notifications sur les plates-formes Windows, Android et iOS.

Cette rencontre sera l'occasion de discuter et d'échanger autour des nouveautés de Microsoft. Un pot de l'amitié sera offert par l'école afin de passer un moment convivial.

février

NI Days

Le 3 février, National Instruments organise sa journée technique à Paris. De multiples conférences et sessions techniques seront proposées autour des outils de l'éditeur. Les concours robotiques seront eux aussi au rendez-vous !
Site : http://france.ni.com/nidays

TechDays 2015

Les 10, 11 et 12 février, Microsoft organise son événement annuel : les TechDays. Comme chaque année, plus de 300 sessions seront jouées sur les 3 jours. Le grand thème de cette édition sera l' « ambient Intelligence » avec la mobilité, le big data, le cloud, le machine learning, les objets connectés. Bien entendu, DevOps sera présent ainsi que les outils de développement.

Pour en savoir plus :
https://techdays.microsoft.fr

Quelques dates à retenir

Le Paris JUG annonce pour début 2015, les soirées suivantes :
- 12 janvier : soirée Young Blood II
- 10 février : soirée Cassandra
- 9 mars : thème ouvert

site : http://www.parisjug.org

Le Breizh JUG annonce :
8 janvier : soirée JHipster (générateur de code) site : http://www.breizhjug.org

écoconception Web (suite)
Serveurs : éviter les goulots d'étranglement

Côté serveur, un code efficient est avant tout un code qui ne bloque pas les autres traitements et libère rapidement les ressources du serveur physique.

Frédéric Bordage, *expert écoconception logicielle*, GreenIT.fr, @greenit
Jérémy Chatard, *directeur technique de Breek*, @breekfr

La base de données et le serveur d'application forment souvent le principal goulot d'étranglement d'une architecture Web. Chaque opération qui prend trop de temps induit un temps latence supplémentaire avant le traitement suivant. A ce niveau, l'écoconception logicielle consiste donc surtout à écrire un code efficient afin de garantir un temps de traitement le plus court possible, pour libérer au plus vite le serveur afin qu'il soit disponible pour un nouveau traitement. On économise ainsi des ressources (nombre de machines physiques nécessaires pour délivrer le service).
Pour garantir qu'aucun goulot d'étranglement n'imposera d'ajouter des machines supplémentaires, il faut partir du fond de l'architecture et la remonter progressivement. En effet, plus le blocage est loin du navigateur dans la chaîne applicative et plus il la paralysera dans son ensemble. Commençons donc par la base de données.

Limiter les requêtes SQL au strict minimum #55, #57

Le phénomène d'obésiciel (ou « gras numérique ») n'est pas limité au code qui s'exécute sur le serveur d'applications ; les requêtes SQL sont souvent les premières concernées car les développeurs sont rarement des as du SQL. Il faut pourtant les optimiser dès leur écriture, et non en fin de projet (car on ne le fait jamais). La première recommandation, qui semble une évidence, mais qui n'est jamais mise en œuvre sur le terrain, est de ne jamais écrire de requête du type `SELECT * FROM`. En effet, le serveur de base de données doit résoudre les champs en fonction du schéma. Si vous connaissez le schéma, nommez les champs.
Par exemple, ne pas écrire :

```
SELECT * FROM clients;
```

Préférez :

```
SELECT raison_sociale, adresse, code_postal, telephone FROM clients;
```

La seconde bonne pratique consiste à limiter le nombre de résultats retournés par la base de données. Rappelez-vous que chaque donnée inutile provenant de la base de données sera traitée par le serveur d'application, transportée sur le réseau, retraitée par le navigateur, etc. C'est donc en début de chaîne applicative qu'il faut limiter les dégâts. Si vous souhaitez n'afficher que les 10 premiers enregistrements d'une table contenant le nom et le prénom de personnes, lors de la sélection, utilisez la clause `LIMIT`. Par exemple, remplacer :

```
SELECT prenom, nom FROM personnes;
```

Par :

```
SELECT prenom, nom FROM personnes LIMIT 0, 10;
```

Utiliser le SGBD/R pour ce qu'il sait faire #55, #56, #58

Lorsque cette fonctionnalité est disponible et que vous ne recourez pas à un middleware qui gère l'accès aux données pour vous, commencez par utiliser les procédures stockées. Les procédures stockées sont bien plus performantes que les requêtes SQL envoyées par le serveur d'application car les requêtes SQL sont déjà compilées. Il faut donc s'appuyer au maximum sur cette fonctionnalité de la base de données pour réduire les cycles CPU et la mémoire vivre consommée par le SGBD/R. Evitez également d'effectuer des requêtes SQL à l'intérieur d'une boucle car cela pose de gros problèmes de performance. En effet, les serveurs SQL sont optimisés pour traiter plusieurs sélections, insertions ou modifications dans une seule requête ou une seule transaction. Envoyer un grand nombre de petites requêtes constitue le meilleur moyen de mettre le serveur de base de données à genoux.
Par exemple, éviter d'écrire :

```
foreach ($userList as $user) {
  $query = 'INSERT INTO users (first_name,last_name) VALUES
("'' . $user['first_name'] . '", "' . $user['last_name'] . '")';
  mysql_query($query);
}
```

Ecrire plutôt :

```
$userData = array();
foreach ($userList as $user) {
  $userData[] = '("'' . $user['first_name'] . '", "' . $user
['last_name'] . '")';
}
$query = 'INSERT INTO users (first_name,last_name) VALUES'
 . implode(',', $userData);
mysql_query($query);
```

Enfin, le meilleur moyen de réduire la charge du SGBD/R est de ne pas l'utiliser. Déclencher une connexion à un serveur de base de données est coûteux en ressources pour l'applicatif, pour le serveur de base, éventuellement pour le réseau. Chaque fois que l'applicatif peut se passer de la base de données, faites-le. Ce conseil peut paraître évident, mais la segmentation des rôles et des responsabilités dans les équipes de développement nous amène à rencontrer ce type d'absurdité à chaque audit que nous réalisons. Bien souvent, les développeurs ne disposent pas d'assez de temps pour optimiser leur code, ils préfèrent donc interroger la base de données aussi souvent que nécessaire, sans chercher à stocker le résultat des requêtes dans des caches ou des variables locales. C'est pourtant la méthode la plus efficace pour décharger le SGBD/R.

Réduire la quantité de mémoire consommée par le serveur d'applications #48, #49

Utiliser des variables statiques. Quand c'est possible, l'utilisation de variables statiques limite le gaspillage de temps CPU en évitant d'exécuter plusieurs fois le même code. Cette technique est particulièrement utile pour les procédures gourmandes en ressource.
Par exemple, pour ne charger qu'une seule fois un parser lourd à initialiser, utiliser les variables statiques :

Dell Precision

Les stations de travail de référence : performantes, innovantes, évolutives

Fixes, mobiles ou montées en rack, les Dell Precision intègrent un ensemble unique d'innovations et de services à forte valeur ajoutée qui en font aujourd'hui les stations de référence sur le marché.

Avec près de 15 modèles soit une largeur de gamme unique, les stations de travail Dell Precision couvrent les besoins les plus variés : ingénierie, architecture, développement informatique, conception de sites marchands, modélisation financière, post-production audiovisuelle. Notons que ces stations offrent la possibilité d'avoir plusieurs machines virtuelles dans un PC de format compact. Chaque station (mobile, tour ou rack) fait l'objet d'un soin très attentif dans ses composants : cartes mères, processeurs graphiques, mémoire vive, disques durs. Ainsi le modèle Tower 7910 équipé de deux processeurs Intel Xeon E5-2670 v3 (2,3 GHz Turbo, 12 cœurs, 30 Mo de mémoire cache), avec carte graphique Nvidia Quadro K5200, 64 Go de mémoire DDR4-2133 et disque dur SSD de 256 Go est une des stations des plus performantes au monde (et on est encore loin du plafond, puisque l'ont peut aussi opter pour une carte graphique K6000, des processeurs plus rapides, jusqu'à 1 Téra de RAM et 8 disques durs). Elle est adaptée aux tâches à haute intensité comme la modélisation 3D avec support des écrans 4K, en particulier avec la solution Catia de Dassault Systèmes, lequel fait partie des logiciels certifiés.

Une sélection très rigoureuse

Les Dell Precision présentent des caractéristiques techniques et fonctionnelles similaires aux serveurs d'entreprise. Les processeurs sont de type Intel Xeon, Intel Core i7 ou Intel Core I5 et peuvent embarquer jusqu'à 18 cœurs avec possibilité de sur cadencement (overclocking) à l'aide de la technologie TurboBoost, sans conséquence thermique négative. La mémoire vive est de type DDR4, au taux de transfert deux fois plus rapide que la DDR2. A cela s'ajoute la technologie de mémoire fiable (RMT) brevetée Dell qui optimise le temps de disponibilité et élimine les erreurs mémoire. Au total, les modèles portables peuvent accueillir 32 Go de mémoire vive et cette capacité est portée à 1 Téra pour les modèles Tour ou en Rack. De même, des contrôleurs de mémoire sont intégrés dans chaque cœur de processeur. Pour les disques durs, les Dell Precision proposent de nombreuses options de stockage: SSD, SCSI... Jusqu'à 3 disques durs dans certains configurations portables et 8 dans les modèles fixes peuvent être insérés en particulier dans des configurations RAID. Les cartes graphiques sont des modèles Nvidia Quadro ou AMD Fire Pro et les formats racks offrent la possibilité de prendre en charge plusieurs cartes avec jusqu'à 6 Go de mémoire dédiée. Outre la puissance pure qui garantit des performances en constante amélioration, Dell propose également une solution gratuite d'optimisation des performances : DPO (Dell Performance Optimizer). Cette solution logicielle permet de paramétrer la station selon l'application utilisée et d'ameliorer son rendement jusqu'à plus de 40%. Enfin, le support des interfaces tactiles permet de tester en temps réel les applications requérant cette dimension ergonomique.

Intégré dans un écosystème complet

Outre une capacité d'évolution unique – par exemple, les blocs d'alimentation peuvent être changés depuis l'extérieur – les stations de travail Dell Precision bénéficient d'un programme de certification de plus de 60 logiciels métiers élaboré avec les ingénieurs de Dell, les éditeurs de logiciels et les concepteurs des composants. Des tests rigoureux sont réalisés pour que l'ensemble matériel & logiciel soit opérationnel dès la mise en service. Les Dell Precision bénéficient du support Pro disponible 24H/24 et 7j/7 qui garantit une intervention d'un technicien expert en moins de 4 heures en cas de problème. Dès lors qu'une entreprise éprouve le besoin de disposer d'un outil performant capable de répondre dans des délais très courts, les stations de travail Dell Precision apportent la réponse la plus efficace en termes de performances, de fiabilité et d'évolutivité préservant ainsi au mieux l'investissement réalisé dans l'outil informatique.

```
private function dataParserLoad() {
  static $done = FALSE;
  if (!$done) {
    require_once APPLICATION_PATH_APP . '/libraries/DataParser/
Parser.php';
    $this->parser = new DataParser();
    $done = TRUE;
  }
}
```

Libérer de la mémoire les variables qui ne sont plus nécessaires. Supprimez notamment les tableaux qui peuvent rapidement contenir des quantités importantes d'informations.
Par exemple, optimiser le code suivant :

```
$crm = new CrmConnection();
$this_month_clients = $crm->fetchAllClients()->filtersByLast
Activity(LAST_MONTH);
$recipes = some_long_function_which_extracts_recipes_from_
clients($this_month_clients);
$pdfs = generate_pdf_for_recipes($recipes);
return $pdfs;
```

Par :

```
$crm = new CrmConnection();
$this_month_clients = $crm->fetchAllClients()->filtersByLast
Activity(LAST_MONTH);
unset($crm);
$recipes = some_long_function_which_extracts_recipes_from_
clients($this_month_clients);
$pdfs = generate_pdf_for_recipes($recipes);
unset($recipes);
return $pdfs;
```

Evidemment, le plus simple est de ne pas assigner inutilement de valeur aux variables. Eviter de déclarer et d'utiliser des variables lorsque ce n'est pas indispensable. En effet, à chaque allocation correspond de la RAM occupée. Par exemple, évitez :

```
$clients = $crm->fetchAllClients();
return $clients;
```

Ecrivez plutôt :

```
return $crm->fetchAllClients();
```

Optimiser le code #47, #50, #62, #63

Comme pour les requêtes SQL, il est préférable de ne pas appeler de fonction dans la déclaration d'une boucle de type for, afin d'éviter que la fonction ne soit appelée à chaque itération de la boucle.
Par exemple, ne pas écrire :

```
for ($i = 0; $i < count($array); $i++) {}
```

Mais :

```
$count = count($array);
for ($i = 0; $i < $count; $i++) {}
```

Utiliser la simple côte (') au lieu du guillemet ("). Pour déclarer une chaîne de caractères en PHP, on peut l'encadrer par des quotes (') ou des guillemets ("). La forme avec les guillemets permet au développeur d'insérer des variables qui seront substituées lors de l'exécution. Si vous n'avez pas de variable dans une chaîne de caractères, utilisez les quotes. Ainsi PHP ne recherchera pas les variables à substituer. Ce qui réduira la consommation de CPU.
Par exemple, optimiser le code suivant :

```
echo ""Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt."";
```

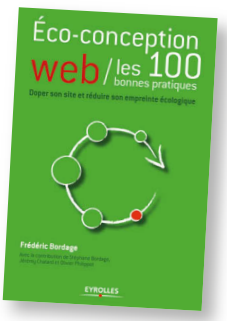
Par :

```
echo 'Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt.';
```

Remplacer les $i++ par ++$i. La forme $i++ a l'inconvénient de générer une variable temporaire lors de l'incrémentation, ce qui n'est pas le cas dans le forme ++$i. On économise ainsi de la mémoire et quelques cycles CPU. Par exemple, $i++ génère 4 opcodes (PHP) alors que ++$i n'en génère que 3.
Mettre en cache les données calculées souvent utilisées. Lorsque des calculs de valeurs ou de données sont coûteux en ressource, les mettre en cache afin d'éviter de recommencer ces opérations alors que les valeurs sont inchangées. Par exemple, les systèmes de cache de type Key-Value Store sont prévus pour stocker ces données. Généralement montés uniquement en RAM, ils génèrent d'importantes économies de CPU si les données calculées sont très souvent sollicitées.
Enfin, si le site repose sur un CMS, ce qui est désormais le cas le plus fréquent, utilisez tous les niveaux de cache du CMS.

Ne pas surcharger le serveur inutilement #46, #52, #65

Dans la mesure du possible, lorsque qu'une ressource est introuvable, ne générez pas de page 404 depuis le serveur d'application et laissez le serveur HTTP (ou le cache) et le navigateur se débrouiller. Lorsque le navigateur demande une ressource qui n'existe pas (image, CSS, fichier JavaScript, etc.), le serveur répond par la page 404. Celle-ci peut être plus lourde que la ressource demandée et donc impacter significativement le serveur d'application. Certains CMS exécutent par exemple leur routine de recherche de contenu (dans la base de données) pour tenter de trouver la page demandée.
Par conséquent, du code serveur est exécuté, le serveur de base de données est sollicité, la génération dynamique de la page HTML est exécutée.
Ce qui aboutit à un gaspillage de CPU, RAM, bande passante.
Dans le même esprit évitez les redirections et utilisez la méthode GET pour les requêtes Ajax. Les requêtes en POST utilisent 2 connexions TCP - une pour le header, l'autre pour les données à transférer – alors que les requêtes HTTP en GET n'utilisent qu'une seule requête TCP. Elles permettent donc une économie en nombre de requêtes.
Enfin, réduisez au maximum la taille des cookies. Ces petits fichiers texte servent à maintenir une connexion entre le navigateur et le serveur HTTP. Le protocole HTTP n'a, en effet, pas été conçu pour des échanges synchrones dans une seule session. Le cookie est donc transféré à chaque requête HTTP. Plus sa taille est réduite et plus vous économisez de bande passante. Dès que la présence d'un cookie n'est plus obligatoire, supprimez-le.



Retrouver ces bonnes pratiques dans le livre : Eco-conception web

ikoula
HÉBERGEUR CLOUD

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

Ordinateurs quantiques : passez à la pratique !

Découvrez comment fonctionnent les ordinateurs quantiques, comment concevoir des algorithmes quantiques, les simuler puis les lancer sur un véritable ordinateur quantique ! L'informatique quantique va prochainement et durablement affecter la sécurité des systèmes cryptographiques et il va falloir profondément revoir nos algorithmes actuels.

Renaud Lifchitz,
consultant sécurité senior à **Oppida**
renaud.lifchitz@oppida.fr

Bases de la physique quantique

Plusieurs règles étranges gouvernent la physique quantique :

- Les objets physiques de petite taille (molécule et taille inférieure) comme les atômes, les photons, les électrons ou les ions, se comportent à la fois comme des particules et comme des ondes lorsqu'on les mesure (principe de dualité),
- Les caractéristiques physiques de ces objets (position, vitesse, polarisation, spin...) ne sont pas déterminées lors de leur création, mais sont en réalité des superpositions de valeurs données par des ondes de probabilité (principes de superposition quantique et d'incertitude d'Heisenberg),
- Une interaction d'un de ces objets avec un autre (ou une mesure) va faire effondrer son onde de probabilité en un unique endroit, déterminant une mesure physique unique et stable (principe de décohérence quantique),
- Par conséquence, copier l'état quantique d'un objet n'est pas possible (théorème de non-clonage).

Nous pouvons tirer parti des trois premiers principes pour réaliser de puissants algorithmes inexistants sur les ordinateurs classiques. Nous stockerons nos informations non pas sur des bits classiques, mais sur des bits quantiques, appelés qubits. Un qubit sera porté par la caractéristique d'un objet physique de petite taille, par exemple la polarisation d'un photon. Si lors de la mesure ce photon est polarisé verticalement, notre qubit porte la valeur 1 (notée |1>), s'il est polarisé horizontalement, la valeur 0 (notée |0>). Evidemment, il existe des états intermédiaires, par exemple une polarisation à 45 degrés. Dans ce cas précis, si l'on mesure le photon avec un filtre droit, il apparaîtra une fois sur deux horizontal, et une fois sur deux vertical, ce qui nous permet de coder une superposition égale d'état 0 et 1. En travaillant avec des qubits « mixés » de ce type, nous pourrons exécuter des algorithmes sur toutes les entrées numériques possibles en les exécutant une unique fois ! C'est ce qui fait la puissance phénoménale des ordinateurs quantiques.

Type	Basis	U	Name	Sym
Pauli	{|0⟩, |1⟩}	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$	X	X
	{|0⟩, |1⟩}	$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$	Y	Y
Z Rotation	{|0⟩, |1⟩}	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$	Z	Z
$e^{i\pi/2}$	{|0⟩, |1⟩}	$\begin{bmatrix}1 & 0\\0 & i\end{bmatrix}$	S	S
	{|0⟩, |1⟩}	$\begin{bmatrix}1 & 0\\0 & e^{i\pi/4}\end{bmatrix}$	T	T
	{|0⟩, |1⟩}	$\begin{bmatrix}1 & 0\\0 & e^{i\pi/8}\end{bmatrix}$	R4	R4
Identity	{|0⟩, |1⟩}	$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$	I	I
Hadamard	{|0⟩, |1⟩}	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$	H	H

Type	Basis	U	Name	Sym
Controlled Not	{|00⟩, |01⟩, |10⟩, |11⟩}	$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$	CNOT (CX)	
	{|00⟩, |01⟩, |10⟩, |11⟩}	$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}$	SWAP	
Measure	{|0⟩, |1⟩}	Qubit to Bit	M	
Binary Control	{|0⟩, |1⟩}	Conditional Application	BC	X
Restore	{|0⟩, |1⟩}	Bit to Qubit	Reset	|0⟩

Tableau des portes quantiques classiques (source Microsoft Research, 2014) **Fig.1**

Aussi étranges que certains principes puissent paraître, cela fait plus de 30 ans qu'ils sont confirmés expérimentalement et que la théorie prévoit des comportements inattendus. Ces principes sont même correctement maîtrisés au niveau industriel et il existe des périphériques informatiques permettant la génération de véritables nombres aléatoires ou encore le chiffrement transparent de flux réseaux par cryptographie quantique, par exemple ceux commercialisés par la société suisse Id Quantique. Ces systèmes sont utilisés quotidiennement par certains réseaux de banques suisses. La cryptographie quantique, si elle est correctement implémentée, a même été démontrée inviolable, y compris par des ordinateurs quantiques !

Portes et circuits quantiques

De façon similaire à nos portes logiques classiques (AND, OR, XOR, NOT,...), il existe une grande variété de portes quantiques agissant sur des qubits. Comme les qubits doivent rester en état de superposition (appelé « état cohérent ») pendant un calcul, il ne doit pas y avoir d'échange de matière ou d'énergie entre le milieu de calcul, et l'environnement extérieur. Une porte quantique ne consomme donc théoriquement aucune énergie et ne rejette donc rien ! Pour ces raisons thermodynamiques aussi, une porte quantique doit être réversible, c'est-à-dire qu'à partir de sa sortie, on doit pouvoir prévoir son entrée. Il s'ensuit qu'une porte quantique AND similaire à la porte logique que nous connaissons déjà, ne peut pas exister, une sortie à FAUX ne permettant pas de savoir quelle est l'entrée correspondante. En conséquence, les portes quantiques ont toutes autant de qubits d'entrée que de qubits de sortie.

Mathématiquement, chaque porte quantique à n qubits peut être représentée par une matrice unitaire de dimension 2^n x 2^n (Figure 1, colonne U). Appliquer une porte quantique à un qubit ou à un ensemble de qubits (appelé « registre ») revient simplement à multiplier à gauche le vecteur colonne de ces qubits par la matrice opérateur de la porte. Combiner plusieurs portes revient à multiplier ensemble leur matrice **Fig.1**.

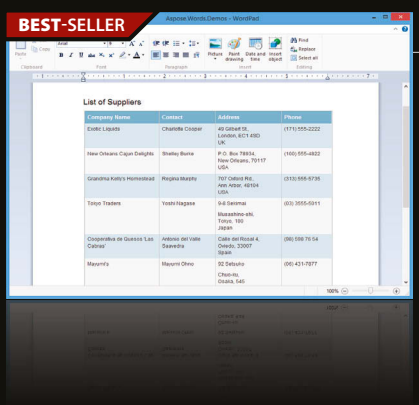
Les portes fondamentales sont : la porte Pauli-X qui effectue la négation d'un qubit (équivalent quantique de la porte classique NOT), la porte d'Hadamard qui permet de mettre en état de superposition équiprobable un qubit |0> ou |1>, la porte Controlled Not (CNOT) qui permet de faire la négation de son

ComponentOne Studio Enterprise 2014 v3 | à partir de € 1 175

ComponentOne a division of GrapeCity

Outils pour le Développeur Professionnel .NET : Windows, HTML5/Web et XAML.

- Contrôles d'interface utilisateur pour .NET, incluant grilles, graphiques, rapports et planificateurs
- Une ligne de HTML5 et JavaScript produits pour le développement d'applications d'entreprise
- Construit en thèmes et un réseau de designers pour créer des thèmes et style personnalisés
- 40+ composants pour Windows 8.1 & Windows Phone 8.1 et support pour Universal Windows
- Supporte toutes les plateformes de Microsoft: Visual Studio 2013, ASP.NET, WinForms, WPF, etc

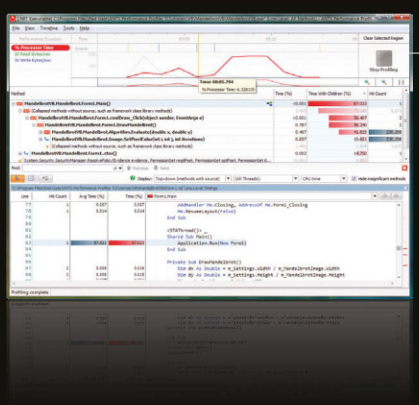


Aspose.Words for .NET | à partir de € 793

ASPOSE
Your File Format APIs

Lisez, modifiez et écrivez des documents Word sans Microsoft Word.

- Création de documents, manipulation du contenu/formatage, puissante capacité de fusion de courrier et exportation en DOC/HTML
- Accès détaillé à tous les éléments d'un document par programmation
- Support les formats de fichiers: DOC, DOCX, WordprocessingML, RTF, HTML, OOXML, OpenDocument, PDF, XPS, EMF et EPUB



Red Gate .NET Developer Bundle | à partir de € 760

redgate®

Éradiquez et corrigez le code lent, identifiez le code .NET bogué et comprenez les raisons.

- Bénéficiez d'une vue d'ensemble des performances de vos applications et identifiez les goulots d'étranglement, dans le code ou la base de données
- Trouvez rapidement les fuites de mémoire et optimisez la mémoire de vos codes C# et VB.NET
- Comprenez et déboguez le code de tiers, frameworks, composants et bibliothèques inclus
- Standardisez la gestion des performances de votre équipe de développement

DevExpress DXperience | à partir de € 1 191

DevExpress

Tous les outils DevExpress ASP.NET, WinForms, Silverlight, WPF et IDE Productivity en un.

- Abonnement de 12 mois pour tous les produits et mises à jour DevExpress et accès aux versions bêta en développement actif
- Modèles et thèmes d'application intégrés exceptionnels
- Support de nouvelle vue Windows 8 UI et panneaux ancrables tactiles
- Support interface codée pour test environnement utilisateur

Siège social en Europe
ComponentSource
30 Greyfriars Road
Reading
Berkshire
RG1 1PE
Royaume-Uni

Siège social aux États-Unis
ComponentSource
650 Claremore Prof Way
Suite 100
Woodstock
GA 30188-5188
États-Unis

Siège social au Japon
ComponentSource
3F Kojimachi Square Bldg
3-3 Kojimachi Chiyoda-ku
Tokyo
Japon
102-0083

Numero vert:
0800 90 92 62
www.componentsource.com

Nous acceptons les bons de commande. Contactez-nous pour demander un compte de crédit.

VISA MasterCard AMERICAN EXPRESS EUROCARD MasterCard

second qubit si le premier est à |1>, et la porte de Toffoli qui permet de faire la négation de son troisième qubit si les deux premiers sont à |1>.

Un ensemble de portes quantiques est appelé « universel » si n'importe quelle opération logique classique peut-être réalisée uniquement par combinaison de plusieurs de ces portes quantiques. La porte de Toffoli seule est universelle, et peut donc réaliser n'importe quel type de calcul, aussi complexe soit-il. Plus couramment on utilise une porte d'Hadamard, une porte de changement de phase (avec phase réglable à Pi/4 ou Pi/2) et une porte CNOT, dont l'ensemble est aussi universel.

On rencontre plusieurs difficultés quand on conçoit un circuit quantique. Tout d'abord les qubits ou registres de qubits ne peuvent absolument pas être copiés, en vertu du théorème de non-clonage. Ensuite, en simulation comme en calcul réel, on doit limiter au maximum l'usage de qubits, cela force donc à les réutiliser dès que possible. Les décalages à gauche ou à droite de registres de qubits, faciles à faire en informatique classique, ne sont pas aisés. Il est bien souvent plus simple de décaler directement les « têtes de lecture » des portes quantiques. C'est ce que nous avons fait pour concevoir l'algorithme de hachage CRC-8 (**Fig.2**), avec seulement 24 portes CNOT ! Enfin, on utilise en pratique des codes correcteurs d'erreurs quantiques pour corriger les erreurs de décohérence qui peuvent avoir lieu pendant le calcul.

Le circuit de calcul quantique de CRC-8 que nous avons obtenu permet le calcul simultané de tous les résultats de CRC-8, en mettant en état de superposition les 8 qubits du registre d'entrée, typiquement avec des portes d'Hadamard. Toutes les portes étant réversibles, le circuit entier l'est aussi, et nous pouvons aussi trouver immédiatement l'entrée correspondante à n'importe quelle sortie, ce qui est notablement intéressant pour une fonction de hachage ! (ceci dit, hachage non cryptographique ici).

Algorithmes quantiques fondamentaux

L'algorithme de Grover est un algorithme purement quantique, probabiliste, itératif et optimal, dont le but est de rechercher parmi N éléments non triés un ou plusieurs éléments spécifiques. Là où en informatique classique il faudrait N tests au pire, et N/2 tests en moyenne, l'algorithme fait mouche en sqrt(N) tests seulement. Conséquence, lors de la recherche d'une clé symétrique de B bits en cryptographie, là où un ordinateur classique doit examiner 2^B cas, l'ordinateur quantique le fait en 2^(B/2) tests, ce qui réduit instantanément toutes les tailles de clés symétriques d'un facteur 2 !

La Transformée de Fourier Quantique (QFT) est l'équivalent quantique de la transformée de Fourier rapide classique (FFT), qui trouve la période dans une superposition de valeurs codées en états.

L'algorithme de Shor, pour la première fois formulé en 1994, permet de factoriser un entier en produit de deux facteurs en temps polynomial et de façon probabiliste, ce qu'on ne sait pas du tout faire avec des algorithmes classiques. Il consiste à trouver exhaustivement les racines carrées non triviales de 1 modulo N et d'en tirer profit pour trouver des facteurs de N, en utilisant la QFT.

D'autres algorithmes fondamentaux existent mais généralement ils ont moins d'intérêt pratique et sont davantage étudiés théoriquement (algorithmes de Deutsch-Jozsa, Simon, ...).

Simulation d'algorithmes et circuits quantiques

Après une conception soigneuse, et avant un test sur matériel réel, il est souhaitable de simuler et déboguer l'algorithme quantique. A cet effet, il existe de nombreux simulateurs. Parmi les simulateurs en ligne sur le Web ne nécessitant pas d'installation, citons « Quantum Circuit Simulator » de Davy Wybiral (http://www.davyw.com/quantum/) ou « Quantum Computing Playground » (http://www.quantumplayground.net/), plus complet, mais plus complexe car disposant d'un langage de script.

Pour les utilisateurs de smartphone Android, citons « Quantum Circuit Simulator » (https://play.google.com/store/apps/details?id=mert.qcs), très accessible et pratique pour tester de petits circuits.

Enfin, le lecteur plus aguerri pourra s'orienter vers un environnement de développement complet comprenant un langage de programmation, à l'instar de QCL (http://tph.tuwien.ac.at/~oemer/qcl.html) ou de l'excellente bibliothèque Python SymPy et de son module « quantum » (http://docs.sympy.org/dev/modules/physics/quantum/). Avec ces outils, le lecteur pourra expérimenter presque sans limite ses propres algorithmes et les algorithmes communs les plus complexes, comme les QFT ou les algorithmes de Grover et de Shor.

Ordinateurs quantiques adiabatiques

La société canadienne D-Wave s'est illustrée il y a quelques années, par l'annonce de la fabrication d'un ordinateur quantique de grande capacité, D-Wave One, dont elle a vendu des exemplaires aux géants américains Lockeed Martin et Google. Cette machine controversée de 512 qubits n'est pas un ordinateur quantique universel et ne permet la résolution partielle que de certains types de problèmes combinatoires basés sur la minimisation de fonctions. Elle a un coût assez prohibitif (10 millions de dollars), nécessite une immense machinerie de refroidissement, et n'a

Fig.2

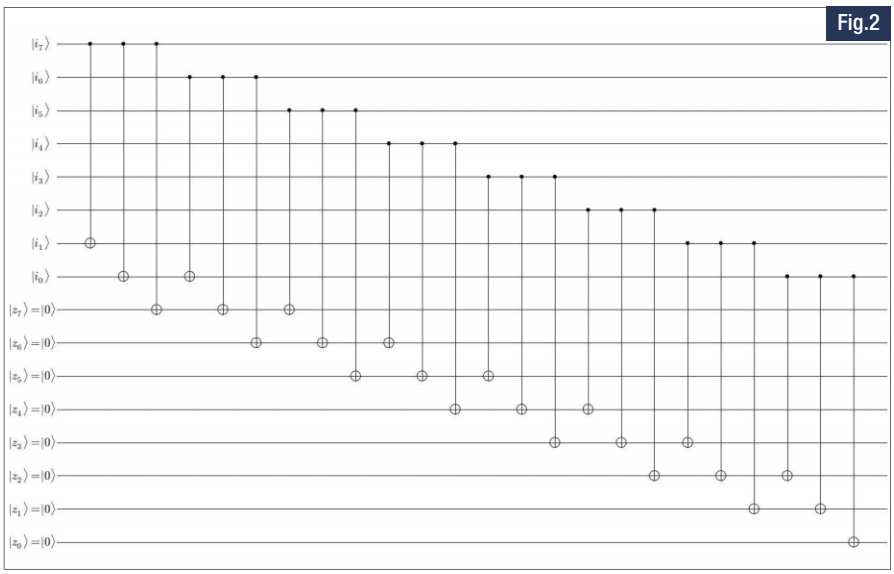


Schéma d'un circuit quantique de calcul de CRC-8

WINDEV

DÉVELOPPEZ 10 FOIS PLUS VITE

Java HTML5 php iOS

Elu
«Langage
le plus productif
du marché»

NOUVELLE VERSION

920 NOUVEAUTÉS

WINDEV AGL N°1 en FRANCE

Fournisseur Officiel de la Préparation Olympique

Tél province: **04.67.032.032**
Tél Paris: **01.48.01.48.88**

www.pcsoft.fr

Des centaines de témoignages sur le site

pas pour l'instant été suffisamment testée par des chercheurs indépendants. D-Wave annonce aujourd'hui une puce de 2048 qubits, et prévoit un doublement de cette capacité tous les ans pendant 10 ans.

Ordinateurs quantiques réels (et universels)

Il est bien plus intéressant de regarder un véritable (et bien réel !) ordinateur quantique, sur lequel n'importe quel algorithme ou circuit peut être implémenté. Le centre de photonique quantique de l'université de Bristol (Royaume-Uni) a en effet lancé l'initiative « Quantum in the Cloud » (http://www.bristol.ac.uk/physics/research/quantum/qcloud/), qui met à disposition de tous, en ligne et gratuitement, une puce quantique optoélectronique (**Fig.3**). Bien que de petite taille (6 fibres optiques en entrée et sortie), celle-ci est fonctionnelle et même universelle pour toute porte quantique à 2 qubits. Les chercheurs de l'université disposent eux, de puces de capacités plus importantes.

Le lecteur curieux et téméraire pourra s'entraîner sur le simulateur en ligne (http://cnotmz.appspot.com/) puis demander un jeton de calcul d'une heure à l'université, après avoir bien compris le paramétrage de la puce pour le calcul et la notion fondamentale de post-sélection (un tutoriel est disponible ici : http://cnotmz.appspot.com/doc/pdf/tutorial.pdf). L'ordinateur quantique est donc bien une réalité !

A la base, cette puce est conçue pour être une porte CNOT un peu améliorée. Nous avons, dans notre étude, réussi à énumérer et simuler tous les paramétrages possibles de cette puce de façon à pouvoir concevoir n'importe quelle porte quantique à 2 qubits, pour la rendre universelle. Nous détaillerons ici, pour le lecteur, uniquement l'implémentation du cas de base d'une addition de deux qubits (qA et qB), sans retenue. Pour rendre les choses plus intéressantes, nous ne ferons pas qu'une seule addition, mais 2 en parallèle, en ajoutant un qubit mixé (superposition de |0> et |1>) à la constante |1>. Remarquons déjà qu'une porte CNOT convient, car elle additionne ses deux entrées et stocke le résultat dans son second qubit de sortie : (qA,qB) -> (qA,qA+qB). Le paramétrage de la puce consiste à choisir dans quelles fibres nous allons injecter des photons. Pour le qubit qA, ce sera la fibre 1 si qA=|0> et la fibre 2 si qA=|1>. Pour le qubit qB, ce sera la fibre 3 si qB=|0> et la fibre 4 si qB=|1>. Pour placer un qubit en état de superposition équiprobable (ici qA), nous le

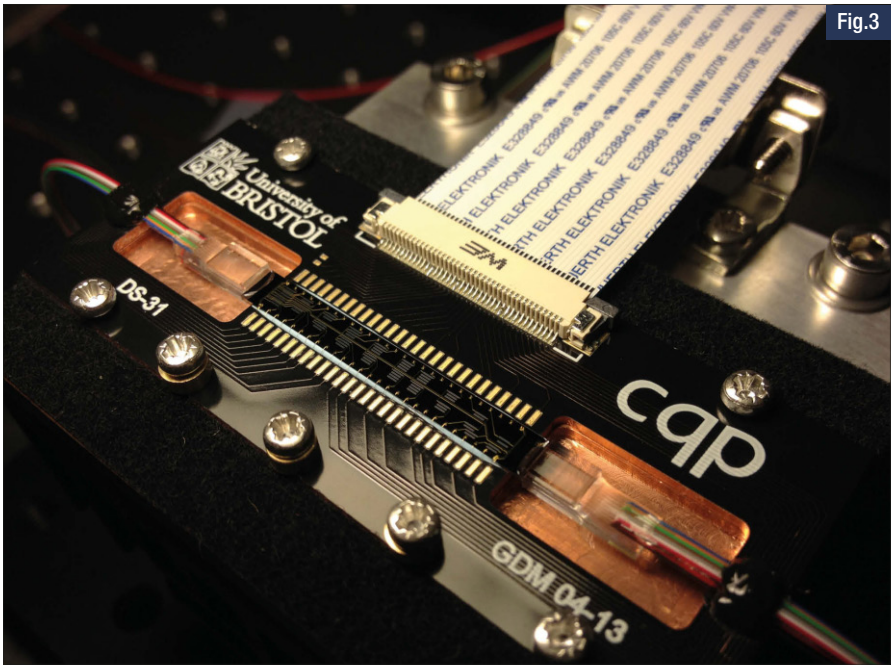


Fig.3

Puce optoélectronique du projet "Quantum in the Cloud"

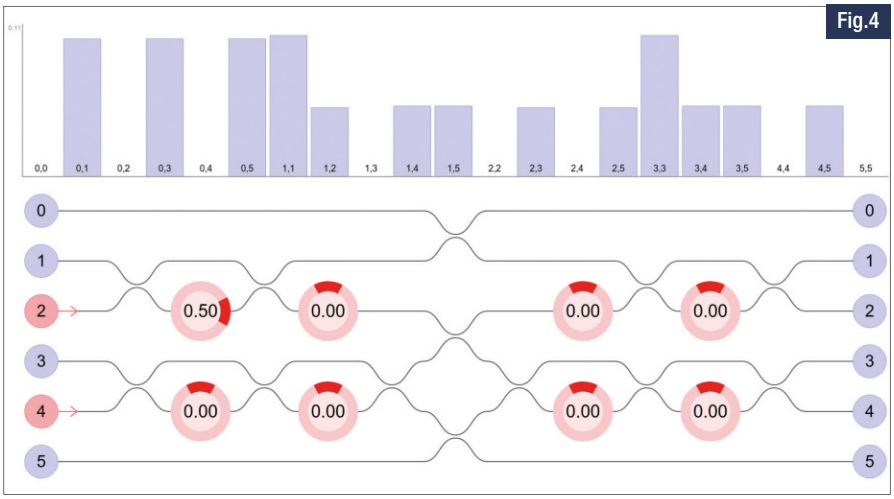


Fig.4

Paramétrage d'une addition quantique

fixons à |1> puis nous paramétrons le changeur de phase de sa fibre sur 0.50 (= Pi/2 radians). Nous obtenons la configuration de la figure 4. L'histogramme en haut de la fenêtre donne les probabilités de détection de photons dans les fibres de sortie. Il faut bien comprendre qu'avec le paramétrage des fibres que nous avons choisi pour qA et qB, détecter un photon en fibre 0 ou 5 ne code aucune information utile. Il faudra uniquement garder les paires de photons résultats dont l'un est en fibre 1 ou 2, et l'autre en fibre 3 ou 4.
Cette étape s'appelle la post-sélection.
Après post-sélection, il ne reste que deux issues possibles : (1,4) et (2,3), qui donnent avec notre convention de codage les paires de qubits (|0>,|1>) et (|1>,|0>).
Nous avons donc réalisé en parallèle les additions |0>+|1>=|1> et |1>+|1>=|0> sur un ordinateur quantique.

Impacts sur la cryptographie et la sécurité

Compte-tenu des avancées techniques de ces dernières années, on peut estimer que d'ici une dizaine d'années, les ordinateurs quantiques auront atteint quelques centaines de qubits, ce qui sera suffisant pour commencer à mettre à mal les algorithmes de cryptographie symétrique. En particulier, comme nous l'avons vu, l'algorithme de Grover a pour conséquence de diviser toutes les tailles de clés symétriques par 2. L'algorithme de chiffrement AES-128, aujourd'hui très utilisé, sera ainsi ramené à une recherche de clé de 64 bits, à peine plus sécurisée que celle de l'algorithme DES, complètement cassé aujourd'hui ! Il faudra donc, à minima, doubler toutes nos tailles de clés symétriques pour rester hors de portée d'attaques quantiques. Il en va de même pour tous les algorithmes de

ABONNEZ-VOUS !

(*) Tarifs France métropolitaine

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT à retourner avec votre règlement à
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

☐ ***Abonnement 1 an au magazine :*** 49 € (au lieu de 65,45 €, prix au numéro)
☐ ***Abonnement 2 ans au magazine :*** 79 € (au lieu de 130,9 €, prix au numéro)
☐ ***Abonnement spécial étudiant 1 ans au magazine :*** 39 €
Photocopie de la carte d'étudiant à joindre
Tarifs France métropolitaine

Offre spéciale : abonnement + clé USB Programmez!
☐ 1 an (11 numéros) + clé USB : 60 €
☐ 2 ans (22 numéros) + clé USB : 90 €
Clé USB contenant tous les numéros de Programmez! depuis le n°100, valeur : 29,90 €
Tarifs France métropolitaine

☐ M. ☐ Mme ☐ Mlle Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'accès aux archives et pour l'envoi d'informations relatives à votre abonnement

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !
☐ Je souhaite régler à réception de facture
** Tarifs France métropolitaine*

PROG 181
***Offre limitée**, valable jusqu'au 31 janvier 2015*

hachage cryptographiques (MD5, SHA1, SHA256, ...) qui sont déjà, pour certains, mis à mal actuellement.

Mais le plus critique concerne la cryptographie asymétrique, telle que nous la connaissons. L'algorithme de Shor permettant de factoriser rapidement les entiers, tous les algorithmes cryptographiques assumant que la factorisation est complexe (de type RSA) sont cassés efficacement. Des variantes de cet algorithme permettent de traiter efficacement le problème du logarithme discret, y compris sur les courbes elliptiques. Les échanges de clés de type Diffie-Hellman sont donc aussi menacés.

L'impact est considérable, ce sont donc la plupart des protocoles de sécurité que nous utilisons au quotidien (PKI, SSL, SSH, HTTPS, sécurité des cartes à puces, ...) qui seront à revoir d'ici environ 25 ans, s'accordent à dire bon nombre d'experts, le temps qu'on atteigne quelques centaines de milliers de qubits exploitables. Il n'y aura donc plus d'authentification, de chiffrement ou de signature électronique actuelle efficace.

Les records officiels de factorisation de clés RSA sont pour l'instant assez faibles : 21 (= 3*7) en Octobre 2012 avec un véritable calcul quantique, et 143 (= 11*13) en calcul adiabatique, essentiellement parce que nous ne disposons à l'heure actuelle que de peu de qubits, et que certains sont en plus utilisés pour faire de la correction d'erreurs.
Maintenant que les premiers résultats concrets sont là et que le domaine de recherche est en pleine effervescence, il est raisonnable de penser que le nombre de qubits des systèmes va doubler tous les 18 ou 24 mois, à l'image de la loi de Moore pour les systèmes informatiques classiques. Certains cryptographes débattent d'ailleurs pour savoir si RSA-2048 sera cassé par un ordinateur classique ou quantique.

Il y a fort à parier que certains services de renseignement sont en avance de quelques années sur le civil, comme le suggèrent les embargos sur certains chercheurs et projets de recherche, aussi il est urgent de se pencher sur des alternatives sécurisées résistantes aux attaques d'ordinateurs quantiques. Ce domaine de recherche est en plein essor et s'appelle la cryptographie post-quantique. Six approches principales sont envisagées (en anglais) : *lattice-based cryptography , multivariate cryptography , hash-based cryptography , code-based cryptography , supersingular elliptic curve isogeny cryptography , symmetric key quantum resistance*. Une conférence sécurité, PQCrypto, est même consacrée à ce domaine d'études depuis 2006. Cependant, ces dernières semaines, certains algorithmes que l'on pensait résistants se sont avérés être attaquables. Il y a donc encore beaucoup de travail pour trouver et prouver qu'il existe des alternatives sûres.
Cet article a fait l'objet d'une présentation par l'auteur à la conférence NoSuchCon 2014 (http://www.nosuchcon.org/talks/2014/D1_05_Renaud_Lifchitz_Quantum_computing.pdf)

Suite du hacking page 22

NIDays

Le 3 février 2015
CNIT Paris la Défense

Vous êtes ingénieur, scientifique ou enseignant et vous souhaitez découvrir les dernières tendances en matière de conception de systèmes, de test et de contrôle ?

Alors NIDays est le rendez-vous à ne pas manquer !

Venez nous rejoindre sur l'édition 2015 de NIDays pour partager les bonnes pratiques de programmation, échanger des informations avec les ingénieurs NI et découvrir les dernières innovations qui inspirent les clients NI et leur permettent de résoudre leurs problèmes d'ingénierie.

- 70 stands de démonstrations
- 2 conférences plénières, avec en invité d'honneur Christophe Galfard, Docteur en physique théorique, spécialiste des trous noirs et de l'origine de l'Univers
- 50 conférences techniques et métiers
- 18 sessions de TP pour s'initier aux matériels et aux logiciels
- 2 coupes de robotique pour l'enseignement

Programme et inscription gratuite sur **www.nidays.fr** 01 57 66 24 24

NATIONAL INSTRUMENTS™

Réalisation d'une configuration PC (très) haut de gamme pour jouer 4K

Cet article est à classer dans la catégorie « gros craquage ». Mais il présente, assez sérieusement, le cheminement qui m'a amené à me construire un PC sous Windows pour profiter au mieux des derniers jeux en 4K.

David Rousset - @davrous
Evangéliste Technique HTML5 et Jeux Vidéo
Co-auteur du moteur de jeu WebGL babylon.js

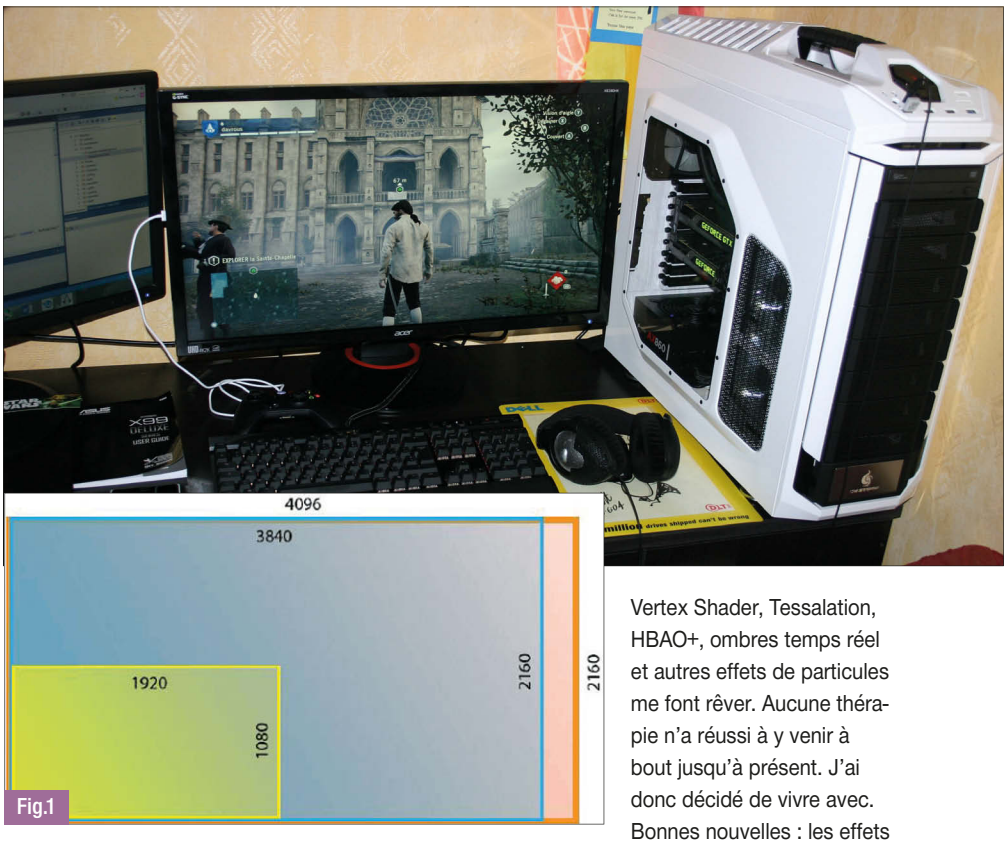


Fig.1

La 4K ?

Le terme 4K englobe en fait 2 résolutions. L'Ultra HD, de 3840x2160, utilisée dans les jeux vidéo et qui sera utilisée par la vidéo dans le futur. La « vraie » 4K en tant que telle, indiquant 4096 pixels de large, plutôt utilisée dans le domaine professionnel dans les cinémas numériques. Mais afin de faire simple, c'est le terme générique 4K qui a été retenu pour désigner l'ensemble. Au passage, 4K dans le monde du jeu vidéo, cela signifie donc 4 fois plus que pixels à générer que le Full HD ! Fig.1.

Ensuite, peut-on réellement voir la 4K ? Bien évidemment ! Je peux pourtant lire fréquemment pas mal d'inepties à ce sujet. Cela dépend uniquement de la taille de l'écran et de la distance à laquelle vous vous trouvez de celui-ci. L'œil n'a pas de résolution "maximum". Par contre, l'œil a bien une limite physiologique. Si vous vous trouvez trop loin d'un écran trop petit, vous ne serez plus en mesure de faire la différence entre un écran Full HD et Ultra HD. Tout est très bien résumé dans cet article : http://www.lemondequitourne.fr/leblog/distance-ecran-oeil/ qui explique le pouvoir séparateur de l'œil. C'est la capacité de l'œil à faire la distinction entre 2 points. La limite de notre œil se situe lorsque 2 points forment un angle de 1/60°. Ensuite, il ne reste plus qu'à faire un peu de trigonométrie. Dans mon cas, j'ai pris un écran 28 pouces en Ultra HD (3840 pixels de largeur). Calculons alors rapidement la distance de confort pour cet écran en partant de la méthode de calcul fournie par l'article précédent.

Résolution horizontale = **3840 pixels**
Diagonale de l'écran = **28 pouces** = 71,12 cm.
Donc largeur de l'écran (16/9) = 61,99 cm
La distance entre 2 pixels = 61,99 cm / 3840 = 0,01614322916 cm
Au final, la **distance de confort idéale** pour cet écran est de : 0,01614322916 / tan(1/60) = **55,49 cm**

Vous pouvez donc **parfaitement profiter du gain de qualité d'un écran 4K dès 28 pouces**, dès l'instant que vous êtes relativement proche de ce dernier. Pour ma part, c'est exactement mon cas. Je suis dans une configuration de type bureau. L'écran est collé à moi d'environ 50 cm justement. Ensuite, plus vous serez éloigné, plus il vous faudra augmenter la taille de l'écran pour justifier la 4K. Donc, oui, la 4K apporte réellement un gain de qualité parfaitement perceptible à l'œil.

Mais jouer en 4K ? Quel intérêt ? **Chacun trouve du plaisir de manière différente dans les jeux vidéo**. Pour moi, la qualité des environnements est importante. Il m'arrive souvent, même dans des jeux d'actions frénétiques, de m'arrêter et de contempler la qualité de la modélisation, du travail artistique des artistes 3D, du niveau technique du moteur (et de me faire fragger la tête comme un bleu du coup au passage).

Bien sûr, le gameplay prime avant tout. Un jeu magnifique en 4K avec un gameplay nul, restera un jeu beau mais chiant... et donc inutile. Malgré tout, je suis un fana du beau pixel depuis mon tout jeune âge. Pixel Shader, Vertex Shader, Tessalation, HBAO+, ombres temps réel et autres effets de particules me font rêver. Aucune thérapie n'a réussi à y venir à bout jusqu'à présent. J'ai donc décidé de vivre avec. Bonnes nouvelles : les effets graphiques disponibles sur PC sont bien plus avancés que sur console.

Pourquoi monter en résolution sinon ? Simplement pour les mêmes raisons qu'à fluidité et effets constants, un jeu est plus beau en 1440p qu'en 1080p lui-même au-dessus du 720p, etc. Bien sûr, il faut aussi que le jeu et son moteur suivent cette montée en résolution en fournissant des textures et effets adaptés. Et je vous confirme que l'expérience est incroyable ! Je joue désormais à Assassin's Creed Unity, Far Cry 4 ou Tomb Raider en 4K. C'est juste bluffant. C'est même toujours aussi étonnant à la vitesse à laquelle on s'habitue à la montée en qualité. Revoir le même jeu en 1080p le rend beaucoup moins précis alors qu'il est déjà magnifique dans cette résolution. Mais franchement la différente entre 4K et 1080p est nette.

Cependant, il est important de noter que **la qualité du rendu (effets graphiques) et la fluidité prime sur la résolution**. Sur une console par exemple, il est nettement préférable de baisser la résolution de rendu en dessous de 1080p pour activer des effets supplémentaires

plutôt que de faire bêtement du 1080p avec un rendu global inférieur. Le rendu pur 1080p (ou 4K d'ailleurs) est donc un faux débat. Nous sommes plus sensibles à la qualité globale du rendu qu'à la finesse des pixels. C'est acquis. Par ailleurs, il faut une **fluidité comprise entre minimum 40 et 60 fps pour satisfaire notre cerveau et notre œil**.

Je me suis d'ailleurs longtemps posé cette question : « quel est le FPS optimal pour l'œil humain ? ». C'est un sujet bien plus complexe qu'il n'y parait. Et ceux qui pensent que le cinéma est à 24fps pour cette raison ont tort. Le meilleur article que j'ai trouvé abordant ce sujet est celui-ci : Framerate et FPS : l'éternelle question enfin résolue (enfin, presque). Et dans le domaine du jeu, on considère souvent un jeu fluide à partir de 30fps et parfaitement fluide à 60fps et au-delà.

Donc, quand j'ai commencé à réfléchir à la possibilité de monter une config PC 4K avec tous ces paramètres en tête, je n'étais pas sûr que cela fût véritablement techniquement possible. **Jouer en 4K dans de bonnes conditions (effets & fluidité), un délire ?** Pas forcément. Je vais donc vous partager dans cet article les questions que je me suis posées avant de choisir les différentes pièces de la bête pour y arriver.

Attention cependant. J'ai vraiment décidé de me faire plaisir. Il serait indécent et stupide de comparer ma superbe configuration à une Xbox One par exemple. Il y a un rapport de 1 à 10 au niveau du prix. J'ai bien conscience que le budget n'est pas raisonnable. A noter également que la machine me servira aussi de machine de « boulot » pour coder dans la bonne humeur sous Visual Studio. Oui, la 4K rend le poil du développeur brillant. Il faut le savoir.

ETAPE 1

la partie la plus importante : GPU & moniteur

Tout d'abord, soyons direct : **aucun mono-GPU actuel n'est assez puissant pour encaisser tous les jeux récents en 4K** avec les paramètres à fond de manière fluide. J'ai pu rapidement m'en rendre compte en lisant les différents benchmarks disponibles sur Internet et en testant avec un seul GPU. Et je le rappelle à nouveau : **mieux vaut jouer en 1080p ou 1440p de manière optimale qu'en 4K à la peine**. La solution passe donc forcément par une solution bi-GPU. Oui mais **combien de cartes et quel GPU ?** Fig.2.

En allant sur mon blog, vous retrouverez la liste des articles m'ayant aidé à faire mon choix. J'ai opté pour une solution de double SLI de nVidia GTX 980. En regardant les

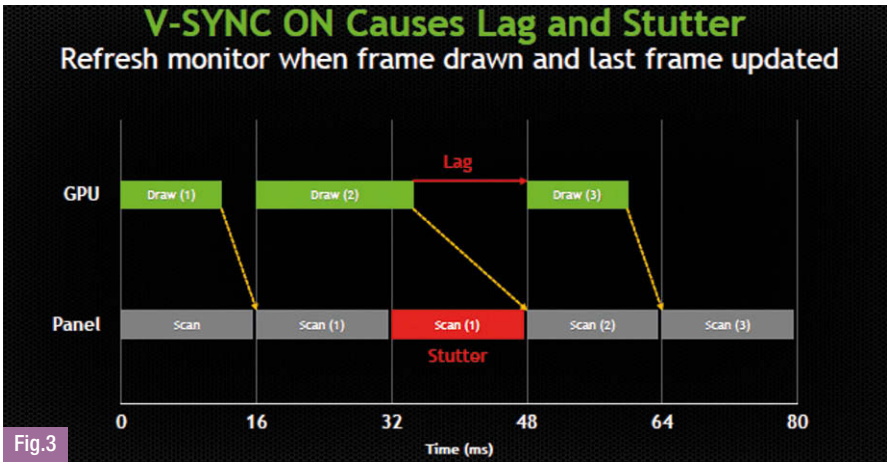


Fig.3

Fig.2

benchmarks, on s'aperçoit aujourd'hui que c'est la solution la plus à même de s'en sortir presque tout le temps. ATI arrive parfois à faire mieux en Cross Fire avec un GPU un peu moins bon comme le 290X. Ils ont donc l'air de mieux maitriser cet art complexe du rendu multi-GPUs. Mais après avoir écumé tous les tests de la planète, j'ai été convaincu par une solution nVidia pour 2 critères principaux :

1 - **L'architecture Maxwell plus moderne**, nettement moins énergivore et donc plus silencieuse

2 – G-Sync comme je vais y revenir un peu plus loin

Par ailleurs pour le même prix, je me suis posé la question de 2 x 980 ou 3 x 970 ? En effet, la GTX 970 est certainement le meilleur rapport qualité/prix du moment. Sauf que, comme vous pouvez le voir dans les articles partagés sur mon blog, le scaling de performance marche désormais très bien en double carte (gain de souvent 60% et plus) mais s'écroule en triple et encore plus en quad SLI.

Mieux vaut donc envisager d'overclocker une solution double SLI que d'investir dans une 3ème carte qui apportera rarement plus de performance. Parfois même pire : il y a des scénarios où l'on perd de la performance avec 3 cartes ! Pour finir, un collègue, ancien ingénieur chez nVidia qui avait bossé sur Fermi (GTX 580) me confirmait via un vieux proverbe g33k « *2 GPUs valent mieux que trois tu l'auras* ». Et j'ai tendance à faire confiance aux magiciens capables de concevoir une puce avec 5 milliards de transistors. ;)

Tout cela est bien joli, mais en jetant à nouveau un œil à tous ces superbes benchmarks, on arrive à une conclusion un peu amère. **Même avec 2 monstres GTX 980, il n'est pas garanti de toujours tenir du 60 fps ou plus en 4K** avec les options à fond aujourd'hui. Alors certes, on peut désactiver certains niveaux d'anti-aliasing qui n'ont plus d'intérêts en 4K et libérer un peu de puissance. Mais malgré tout, il n'est pas rare de se retrouver sur certains jeux entre 40 et 60 fps avec un SLI de 980. N'oublions pas que **4K = 4 fois plus de pixels que le bon vieux 1080p** !

Mince! Si proche du but... Pourtant, entre 40 et 60 fps, on considère l'expérience déjà très bonne pour jouer. Pourquoi est-ce vraiment un problème d'être en-dessous de 60 fps ?

Pour mieux comprendre cela, il faut revenir au fondement de l'affichage vidéo. En fait, cela pose **problème de ne pas être pile poil synchronisé avec la fréquence de rafraichissement verticale** de nos moniteurs. Trop vite (+ de 60 FPS/Hz), le moniteur ne suis pas. La carte envoie une nouvelle image avant que le moniteur ait fini son balayage. Cela entraîne des phénomènes de déchirement ou « tearing » en anglais. Trop lent (- de 60 FPS), le moniteur n'a pas assez d'images à manger et cela engendre des saccades.

Mais pourquoi bon sang devons-nous nous synchroniser sur la fréquence du moniteur ?!? C'est un héritage de l'époque merveilleuse de nos écrans cathodiques. Non content de nous avoir flingués le dos pendant des années, ils hantent aujourd'hui les technologies LCD modernes en nous imposant des limites inutiles Fig.3.

C'est nVidia qui s'est récemment attaqué au problème avec une solution simple et pourtant évidente. Et si c'était plutôt **nos moniteurs LCD qui se synchronisaient de manière dynamique sur la fréquence de génération d'images par seconde du GPU** ? Tellement

évident que l'on se demande pourquoi cela n'est pas arrivé plus tôt. Cette technologie s'appelle **G-Sync** et permet de rendre les jeux parfaitement fluides dès 40 fps. Mais dis donc, cela fait bien mon affaire ! Et je vous confirme que cette technologie est magique ! Sans G-Sync, mon expérience de jeu serait clairement dégradée aujourd'hui.

ETAPE 2

CPU, mémoire, carte mère et tout le toutim

Les jeux exploitent encore assez mal les **multi-cœurs de nos CPU** mais cela est en train de nettement s'améliorer. Malgré tout, pendant longtemps, il valait mieux privilégier un CPU modeste capable de monter haut en fréquence plutôt que forcément un nombre cœurs plus élevé avec une fréquence réduite. Comme les jeux utilisaient principalement le cœur n°1, il fallait s'assurer que celui-ci était capable de monter le plus haut en fréquence. Cependant, vous l'aurez compris, je monte une configuration haut de gamme censée tenir dans le temps. J'ai donc décidé de me projeter légèrement et de faire un pari, peu risqué, sur l'avenir proche. Je suis convaincu que la donne va très bientôt changer. Laissez-moi vous expliquer.
Tout d'abord, il faut savoir que **plus le jeu est récent plus il bénéficie du nombre de cœurs** avec une limite vers 6 aujourd'hui. Dans certains cas, 8 permettent d'aller un peu plus loin. Cela tombe bien, je ne me monte pas une config 4K pour jouer à de vieux jeux. ;)
Par ailleurs, Windows 10 va bientôt apporter une petite révolution pour apporter un surplus de performance bienvenue. Cette révolution s'appelle **DirectX 12**. Comme vous pouvez le voir sur ce diagramme, **DirectX 12 utilise bien mieux l'architecture multi-cœurs** : Fig.4.

Je pense également que DirectX 12 nous permettra de plus facilement bénéficier du portage de jeux optimisés du monde des consoles.
Bref, j'ai craqué et pris le plus gros CPU actuellement disponible : Intel Haswell-E 5960X avec ses 8 cœurs hyperthreadés. C'est le meilleur du moment, un point c'est tout. Alors certes son rapport performance/prix n'est pas bon. Surtout pour les anciens jeux comme j'ai pu déjà vous le dire. Dans la série **Haswell-E, mieux vaut prendre un 5820K, 2 fois moins cher**, voir même surement un Core i5 classique si le but est uniquement de jouer avec son PC. Dans mon cas, le PC va en fait me servir la plupart du temps à faire autre chose que jouer: coder, encoder des vidéos, traitement du son et calculs 3D. Les 16 cœurs logiques du 5960X me faisaient donc méchamment de l'œil. Je souhaitais aussi profiter de ses 40 lignes PCI-Express qui, couplée au chipset X99, permettaient de viser du 16x/16x sur mes 2 copines GTX 980 en SLI. Avec un seul GPU, le 16X n'a aucun intérêt mais comme on peut le voir dans différents tests en ligne, le 16x/16x permet de gagner en performance sur une configuration en SLI.
Côté mémoire, faisons bref : 16 Go de DDR4. Pas besoin d'aller plus haut dans les jeux aujourd'hui ni dans mon boulot de tous les jours. Côté carte mère : je suis fan d'Asus de la première heure et j'ai toujours fait confiance à leurs cartes mères. J'ai jeté mon dévolu sur **l'Asus X99 Deluxe. Les tests sont unanimes et ses capacités d'overclockling au-dessus de la moyenne** semble-t-il.
Au final, si vous voulez vous monter la même configuration que moi, voici ce qu'il vous faut prendre :

- Intel Core i7 5960X - Extreme Edition le top et bel overclocking au programme
- GeForce GTX 980 - 4 Go X 2
- Asus X99-DELUXE
- Corsair Hydro Series H75 car le water cooling aide à l'overclocking
- Cooler Master CM Storm Stryker car je voulais un boitier spacieux et vitré
- Corsair AX860 Modulaire alimentation 860W platinum pour bien donner à manger aux 2 bébés 980 et au papa 5960X
- Samsung Serie 850 Pro - 512 Go car c'est d'après tous les benchmarks du moment le SSD le plus rapide du marché
- G.Skill Ripjaws 4 Black DDR4 4 x 4 Go 2133 MHz
- Acer Predator XB280HK car c'est le seul 4K et G-Sync du marché
- Corsair Gaming K70 RGB car je voulais un clavier mécanique de gamer. Celui-là est complètement fou comme je voulais : rétro-éclairage de couleurs différentes par touche et un système de macro disponible pour faire des animations sur tout le clavier !

Le prix dépasse donc les 4000€. Mais je vous rassure, il y a moyen de faire un peu moins cher, vers 3000€, en grattant sur le CPU et le SSD, si votre objectif est uniquement de jouer. Mais pour la 4K, difficile de faire l'impasse sur du SLI de 980 aujourd'hui. Sauf si vous voulez jouer avec toutes les options au minimum, ce qui n'aurait aucun intérêt ! La 4K Ultra est donc aujourd'hui bien plus chère que le 1080p Ultra.

ETAPE 3

La bête en images, premiers retours et benchmarks

Alors elle donne quoi cette bécane ? Fig.5.
Pour l'instant, je me suis légèrement lancé dans l'overclocking du CPU et des GPUs. Aujourd'hui, je suis sur un BCLK de 128 Mhz

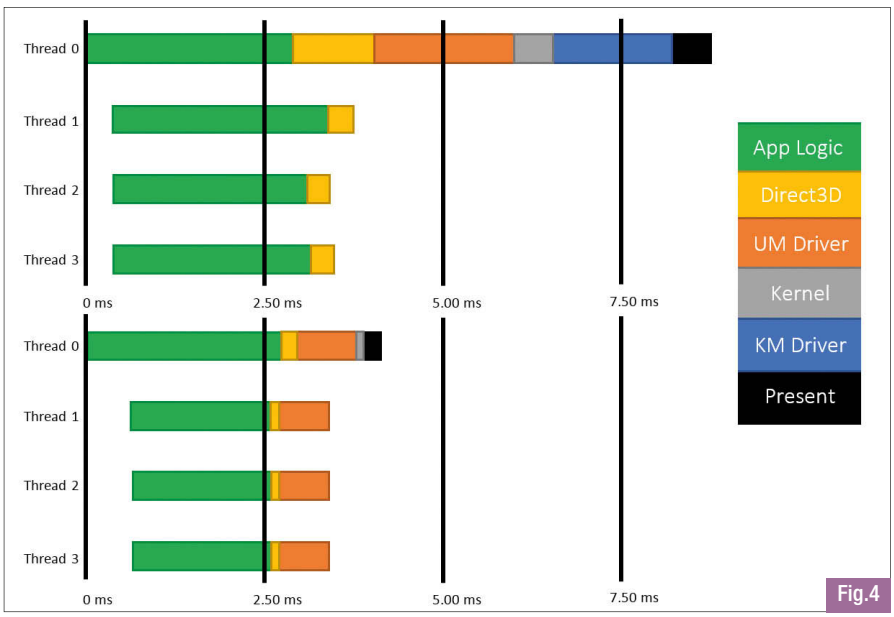


Fig.4

Fig.5

avec un multiplicateur de 33 avec un voltage à 1,275V du CPU soit une fréquence de 4,224 Ghz sur les 8 cœurs. **C'est vraiment un jouet exceptionnel que ce 5960X** ! Les 2 GPUs sont overclockés via l'outil MSI AfterBurner avec +200 Mhz sur le GPU et +200 Mhz sur la mémoire. Cela semble être étonnement stable pour l'instant. Mon fils a réussi à jouer à Far Cry 4 3h d'affilée, c'est plutôt bon signe.
Je vous confirme au passage qu'Assassin's Creed Unity et Far Cry 4 tournent très bien en 4K et en Ultra. C'est juste magnifique ! Par contre, il faut malgré tout faire quelques petites concessions au niveau de quelques options graphiques. Ainsi, pour Far Cry 4, tout est à fond sauf l'anti-aliasing qui est désactivé. Je tourne alors entre 40 et 60 fps, ce qui est parfait pour le G-Sync Fig.6.


Fig.6

Fig.7


Fig.8

Pour Unity, j'ai dû descendre les textures de « Ultra Elevée » à « Elevée », les ombres de « douce » à « elevée » et mettre l'anti-aliasing sur FXAA. Le reste en Ultra et HBAO+. Ainsi, le jeu tourne entre 45 et 60 fps Fig.7.
Du côté de Tomb Raider 2013, pas de question à se poser. Toutes les options sont au maximum ou Ultra (TressFX aussi pour les cheveux) et le jeu tourne entre 45 et plus de 70 fps. Voici ce que donne le benchmark interne du jeu : fps minimum à 46, fps maximum à 70 et moyenne à 58,6 fps Fig.8. Dans ces mêmes conditions, voici ce que cela donne sous **3D Mark 11** en mode performance : Fig.9
J'obtiens la note de **P28553**, ce qui semble plutôt bon en comparaison de la base en ligne. C'est même du niveau de certain triple SLI 980 apparemment !
Note: pensez bien à **désactiver le G-Sync** ! Sinon, le système bloque les FPS à limite maximum de votre écran (60 Hz dans mon cas) et biaise totalement les résultats.
La machine est également particulièrement silencieuse. Inaudible en mode navigation sur Internet, Office et Visual Studio. En mode jeux, les ventilos soufflent un peu plus mais franchement, c'est très faible au niveau du bruit généré. Je suis épaté. Je mets cela sur le compte du gros boitier et de ses gros ventilos, du watercooling et de l'architecture Maxwell particulièrement efficace. Du coup, cela fait tellement peu de bruit que j'ai posé la tour sur le bureau à mes côtés. Voir les 2 petites GTX 980 me procurant un bien fou (n'oubliez pas la pathologie dont je souffre à l'origine avant de vous moquer !).
Pour finir, Windows 8.1 gère parfaitement l'augmentation du DPI du moniteur 4K. C'est apparemment d'ailleurs le meilleur OS pour gérer la 4K aujourd'hui. Je n'ai pas encore pris le temps de tester Windows 10. Côté performance, Windows 8.1 est légèrement plus rapide que Windows 8 lui-même étant optimisé par rapport à Windows 7. Windows 8.1 semble donc être aujourd'hui la meilleure base pour du jeu et travail en 4K. Windows 10 apportera quant à lui, DirectX 12. Mais ça, c'est une autre histoire !

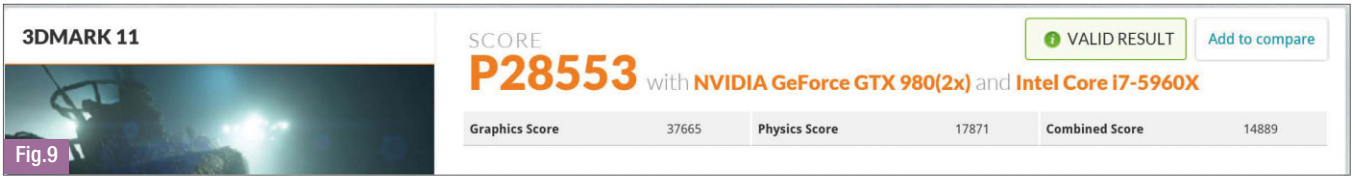


Fig.9

Protégez le code de vos applications Android

Une application Android est composée principalement de fichiers XML, de code Java et de diverses ressources. Tout ceci est transformé par le SDK Android pour être inclus dans un fichier prêt à être distribué. Mais qu'est-il possible de retrouver depuis ce fichier ? Cet article a pour but de vous montrer ce que quelqu'un de bienveillant (ou non) est capable d'extraire de vos applications, ainsi que comment s'en prémunir et utiliser ces techniques lors de votre activité de développeur Android.

Sylvain Galand
Genymobile - *Ingénieur développement*
@sylvaingaland

Comme fil rouge dans cet article, nous allons utiliser AdBlock Plus [1]. L'application AdBlock Plus permet de configurer le proxy HTTP du système et ainsi de pouvoir rediriger et filtrer le trafic réseau.
Dans l'API d'Android, aucun moyen n'est proposé pour configurer le proxy. Comment AdBlock Plus peut-il le faire ? C'est sans doute en utilisant une API cachée, nous allons essayer de retrouver le bout de code qui permet cela via la décompilation.

Qu'est-ce qu'une application Android ?

Du point de vue du développeur, une application Android est composée de fichiers XML, de code Java, ainsi que de plusieurs types de ressources (images, sons, librairies logicielles natives, etc). Cet ensemble de fichiers sera ensuite traité par le SDK Android fourni par Google pour produire un unique fichier APK (Android application Package).
Ce fichier contient l'intégralité du code et des ressources et, une fois signé, pourra être distribué directement ou via les magasins d'applications. Une fois l'application installée, son code sera optimisé pour le système. Mais le fichier APK d'origine sera toujours présent car lors d'une mise à jour majeure d'Android, une passe d'optimisation sur l'ensemble des applications sera nécessaire. Le fait que l'APK soit toujours présent sur l'appareil permet aux utilisateurs de l'extraire très simplement. En activant les options de développeur et notamment ADB (Android Debug Bridge, un outil fourni par le SDK permettant de développer et déboguer sur les systèmes Android), n'importe qui pourra lancer les deux commandes suivantes :

```
# L'option -f permet de connaître le chemin du fichier dans
 le système de fichier Android
$ adb shell pm list packages -f | grep adblock
package:/data/app/org.adblockplus.android-1.apk=org.adblock
plus.android
# Récupérons maintenant l'apk qui est lisible par tout le monde
$ adb pull /data/app/org.adblockplus.android-1.apk
4717 KB/s (3380372 bytes in 0.699s)
$ ls
org.adblockplus.android-1.apk
```

Ceci ne requiert nullement d'avoir des accès privilégiés, les APK sont accessibles par tous les processus, notamment pour que des applications puissent venir utiliser certaines ressources d'autres APK (exemple : les icônes !). Une fois en possession du fichier APK, il est possible de le redistribuer ou de regarder ce qu'il contient.

Récupération des ressources, fichiers XML et du code Smali

Un fichier APK est une archive zip. Il est donc possible de l'ouvrir via n'importe quel logiciel permettant d'ouvrir un zip.

```
$ unzip org.adblockplus.android-1.apk -d unzip
[...]
$ ls unzip/
AndroidManifest.xml  assets/  bin/  classes.dex  lib/
LICENSES  META-INF/  NOTICE  res/  resources.arsc
```

Cependant, les XML sont illisibles car ils sont compilés par le SDK pour une lecture plus rapide lors de l'exécution, et le code est confiné dans un *.dex* (Dalvik Executable). Seules certaines ressources sont accessibles de cette manière. L'outil qui va nous aider à aller un peu plus loin s'appelle *apktool* [2]. Dans le SDK Android, *aapt* est l'outil nécessaire pour transformer le code compilé et toutes les ressources en APK. apktool fait l'inverse.

```
# l'option d est utilisée pour décompiler. En utilisant b,
 on recompile.
$ apktool d org.adblockplus.android-1.apk apktool/
$ ls apktool/
AndroidManifest.xml  apktool.yml  assets  lib  res  smali
```

Il permet de récupérer les ressources, évidemment, mais aussi de rendre l'ensemble des fichiers *XML* lisibles et également de décompiler du *.dex* en *.smali*. Le *smali* pourrait être considéré comme la version désassemblée du *dex*. Même si il n'est pas facilement exploitable, il est tout à fait possible d'y retrouver de l'information ainsi que d'en modifier le code pour reconstruire un *APK*.

```
.method public isChecked()Z
  .locals 1

  .prologue
  .line 102
  iget-boolean v0, p0, Lorg/jraf/android/backport/switch
widget/TwoStatePreference;->mChecked:Z

  return v0
.end method
```

Ci-dessus une méthode extraite du fichier *TwoStatePreference.smali*. Je vous laisse le soin de la décrypter !

Décompilation de .dex en Java

Encore une fois nous allons utiliser un outil pour récupérer directement le code Java, cet outil est *jadx* [3]. *jadx* a l'avantage par rapport à ses concurrents de décoder directement le *.dex* en Java, sans passer par un .jar. Le code Java obtenu est donc de meilleure qualité au final.

```
$ jadx org.adblockplus.android-1.apk -d jadx/
# Il se peut que des erreurs apparaissent le résultat
n'étant pas un code décompilé à 100%
```

[1] AdBlock Plus Android - https://adblockplus.org/fr/android
[2] apktool - https://code.google.com/p/android-apktool/
[3] jadx - https://github.com/skylot/jadx

Voici la fonction que nous avons extraite du *smali*, mais cette fois ci en Java :

```
public boolean isChecked() {
    return this.mChecked;
}
```

Notre quête était de retrouver comment fait AdBlock Plus pour paramétrer le proxy. Une simple recherche du mot “proxy” dans l’ensemble du code remonte une classe intéressante, *ProxySettings.java*. Cette classe ne comprend que cinq méthodes statiques dont :

```
public static boolean setConnectionProxy(Context context,
String host, int port, String excl)
```

En regardant de plus près, cette méthode forge un objet et appelle une méthode privée *setHttpProxy()* par réflexion sur l’objet renvoyé par *getActiveLinkProperties()* du *ConnectivityManager*. Il suffit maintenant de récupérer le code source d’Android pour vérifier tout cela ou de reprendre le code et d’implémenter telle quelle la fonctionnalité dans notre propre application. Nous avons réussi à récupérer le bout de code qui nous intéressait. AdBlock Plus étant open source, il aurait été plus simple d’aller voir directement dans le code mais cela aurait été moins passionnant.
Notons qu’Android propose depuis ses dernières versions une nouvelle machine virtuelle ART, en option pour les développeurs. Nous ne savons pas pour le moment si le .dex va disparaître au profit d’un nouveau format .art et donc si les outils présentés précédemment devront être adaptés.

Comment protéger son code ?

Tous ces outils permettent de retrouver (en partie) le code qui a été utilisé pour compiler l’application. Même s’il n’est pas directement exploitable et compilable, il peut permettre à quelqu’un de mal intentionné de récupérer des parties de votre travail. La valeur du code source, qu’il vienne d’une décompilation ou de votre dépôt de source, est la lisibilité. Si le code est illisible, il n’a aucune valeur. Avoir un code qui compile, mais qu’il est impossible de comprendre et donc de modifier ne vaut pas beaucoup mieux que le binaire. La seule technique efficace contre ce genre d’attaque est donc l’obfuscation. L’obfuscation est une technique permettant de rendre le code illisible notamment en remplaçant les noms de classes, méthodes et paramètres par de simples caractères.

```
package a;

public class a {
    [...]
    public boolean a() {
        return a;
    }
}
```

Notre méthode précédente ainsi que le package et nom de la classe ont été obfusqués. Le code devient donc illisible et inexploitable.
Proguard [4] est la solution d’obfuscation proposée par défaut dans le SDK Android. Proguard va plus loin, car il permet aussi d’optimiser et réduire la taille de votre code en supprimant tous les morceaux de code non utilisés, y compris dans les librairies tierces du projet. L’activation et la configuration de Proguard dépend du système de *build* utilisé [5]. Il faudra bien faire attention à déclarer ce que vous ne voulez pas obfusquer, notamment si vous utilisez de la réflexion, car Proguard n’est pas capable de suivre la réflexion et risque de considérer les méthodes et classes appelées ainsi comme non utilisées. Un fichier *mapping.txt* est généré après l’obfuscation permettant de traduire les éventuelles stacktraces illisibles vers les noms d’origines. Perdre ce fichier reviendrait à ne plus être capable de décoder les stacktraces de vos utilisateurs.
Toutefois l’obfuscation ne permet pas de cacher les appels aux API du framework et peut donc parfois révéler assez d’information pour comprendre ce qu’il se passe.

```
public class a {
  private static String a = "MotDePasseSecurePourChiffrer";
  public static Cipher a() {
    Cipher localCipher = Cipher.getInstance("AES/ECB/PKCS
7Padding", "BC");
    localCipher.init(1, new SecretKeySpec(a.getBytes(), "AES"));
    return localCipher;
  }
}
```

Pour ces problématiques plus précises, il faudra se tourner vers des outils tels que DexGuard, la version payante de Proguard, qui est aussi plus complète.

Quand se protéger ?

Il est important de savoir s’il est utile pour vous de protéger votre code car Proguard demande de la configuration. Il faudra ainsi adapter votre environnement de développement et de suivi pour pouvoir traduire les stacktraces récoltées par vos différents services de remontée d’erreur. Il faudra également passer du temps en QA car les développeurs utiliseront une version non obfusquée différente de la version de release.
Il est important de garder en tête que malgré tout cela, l’obfuscation n’est qu’un obstacle et n’est pas une solution parfaite. N’importe qui avec du temps et des moyens pourra retrouver ce que vous cachez. Évitez de mettre des secrets dans le code de vos applications !

Utilisation de ces outils dans votre environnement de développement

Du point de vue moins obscur de la force, ces outils peuvent aussi aider les développeurs Android dans leur travail quotidien :
Pour vérifier la bonne obfuscation d’un apk avant de le publier. Votre partenaire vous a envoyé son *APK* et une documentation expliquant comment il expose son contenu. Évidemment, la documentation n’est pas correcte et votre contact en vacances, mais vous allez pouvoir ouvrir l’APK pour inspecter et vous débloquer. Auditer en profondeur un *APK*. Récupérer simplement l’*AndroidManifest* d’une application. Avec le même principe, mais des outils différents, il est possible d’aller voir dans le code du *framework* Android propre à chaque constructeur de manière à comprendre pourquoi votre application ne fonctionne pas sur un téléphone en particulier.

Conclusion

Nous avons pu voir qu’il était très simple de récupérer les ressources et le code d’une application Android. Sans être infaillible, Proguard est une bonne solution pour se prémunir de ce problème. Les outils présentés peuvent s’ajouter aux environnements de développement des développeurs Android car ils répondent à des besoins précis. Quant à la question de la légalité, si le piratage est évidemment illégal, la rétro ingénierie dans un contexte d’interopérabilité est légale [7]. La question reste ouverte, à chacun de faire ce qu’il lui semble bon et nécessaire tant que cela sert un intérêt commun. ⊞

Suite du hacking page 95

[4] Proguard - https://www.saikoa.com/proguard
[5] Enabling Proguard - ant - http://developer.android.com/tools/help/proguard.html#enabling
[5-bis] Enabling Proguard - gradle - http://tools.android.com/tech-docs/new-build-system/user-guide#TOC-Running-ProGuard
[6] DexGuard - https://www.saikoa.com/dexguard
[7] Décompilateur - Contraintes légales - http://fr.wikipedia.org/wiki/D%C3%A9compilateur#Contraintes_l.C3.A9gales

Débuter avec Xamarin

Fondée en 2011 par deux anciens collaborateurs (et amis) de Novell, Miguel De Icaza et Nat Friedman, Xamarin a pour objectif de proposer aux développeurs la possibilité de développer sur les plateformes iOS et Android en utilisant les langages C# ou F#, tout en mutualisant un maximum de codes, de logique métier, et jusqu'à l'interface graphique avec Xamarin.Forms.

Thomas LEBRUN – *Consultant*
Infinite Square
http://blogs.infinitesquare.com/b/tom
@thomas_lebrun

Comment ça marche ?

La technologie proposée par Xamarin consiste principalement dans la mise à disposition d'un ensemble de « wrappers » qui permettent aux développeurs d'utiliser les fonctionnalités de la plateforme cible dans leur langage préféré.

Basé sur Mono, Xamarin dispose de deux systèmes de compilation spécifiques, que l'on cible Android ou iOS :

- Une compilation AOT (« *Ahead Of Time* »), pour iOS, permettant de produire un binaire ARM,
- Une compilation JIT (« *Just In Time* »), pour Android, permettant de disposer d'un binaire compilé en langage intermédiaire qui sera ensuite recompilé, à la volée, en natif sur le périphérique (les habitués reconnaitront là un modèle de compilation similaire à ce que l'on retrouve sous .NET). Fig.1.

Il est important de noter que ces deux types de compilations existent pour des raisons bien précises. En effet, de par son architecture similaire à celle de .NET, Android permet l'utilisation de la compilation « *Just In Time* ». Cependant, sous iOS, c'est légèrement différent : Apple n'autorise pas l'exécution de code dynamique sur ses plateformes. Il est donc nécessaire de disposer d'une compilation « *Ahead Of Time* », qui va produire directement un binaire natif qui pourra être déployé et exécuté sur le périphérique.

Cette différence n'est pas sans conséquences car, pour produire ce binaire natif, il est nécessaire de disposer d'une machine (un ordinateur sous OS X en l'occurrence) qui va faire la passerelle entre votre code source et le binaire généré. Grâce à un agent de compilation installé sur la machine, le Xamarin Build Host, votre environnement de développement va être capable de se connecter au Mac, de lui envoyer vos sources et de récupérer, en définitive, un binaire natif.

Que peut-on faire ?

C'est souvent la question que l'on se pose lorsque l'on commence à s'intéresser à Xamarin. Pour y répondre, je recommande toujours de poser la question différemment.

En effet, au lieu de se demander « Que puis-je faire avec Xamarin ? », je préconise de se demander plutôt « Est-ce que la fonctionnalité XXX est possible sous iOS/Android ? ». Si la réponse est oui, alors il sera possible de l'utiliser avec Xamarin.

En effet, les équipes de développement de Xamarin ont fait un travail conséquent et il est possible d'utiliser 100% des APIs disponibles sous Android, iOS et Windows Phone. Ainsi, si une fonctionnalité vous intéresse et si vous souhaitez l'intégrer dans votre application, commencez par vous demander si cette fonctionnalité existe sur la plateforme, et s'il est possible d'y accéder en utilisant les langages natifs (tels que Objective-C / Swift sur iOS et Java sur Android). Si cela est possible, alors vous pourrez l'utiliser avec Xamarin, en utilisant les APIs correspondantes (attention cependant, la difficulté d'implémentation peut varier en fonction de la plateforme).

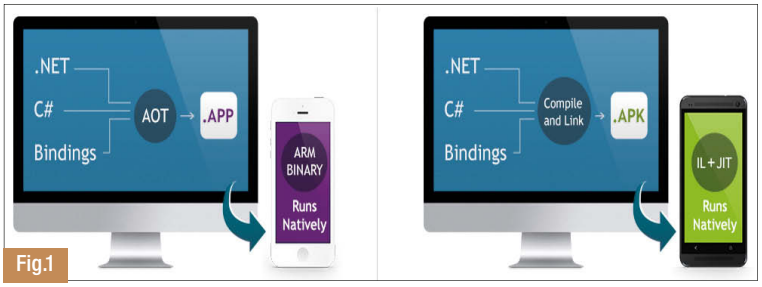


Fig.1

Ce qu'il faut pour démarrer ?

Pour commencer à développer avec Xamarin, il faut télécharger le SDK disponible sur http://xamarin.com/download. Vous pourrez télécharger l'installeur pour Windows ou OS X, en fonction de votre configuration. Attention, en fonction de votre système d'exploitation, vous aurez accès à des fonctionnalités spécifiques (par exemple, sous Mac OS, vous aurez accès à Xamarin.Mac) : Fig.2.

De même, certaines fonctionnalités (comme le support de Visual Studio) ne sont disponibles que dans des versions spécifiques de Xamarin. Ainsi, la version « *Free* » vous permettra de démarrer, de tester, mais ne vous permettra pas de générer des applications dépassant une certaine taille. La licence « *Indie* », idéale pour les développeurs indépendants, intègre en plus Xamarin.Forms mais il faut attendre la version « *Business* » pour pouvoir disposer du support de Visual Studio ainsi que des fonctionnalités « Business » comme WCF (Windows Communication Foundation) par exemple : Fig.3

Pour ne pas bloquer les développeurs, chaque version de Xamarin (de la version « Indie » à la version « Enterprise ») intègre Xamarin Studio, un environnement de développement basé sur MonoDevelop et qui permet de développer des applications pour iOS / Android (en fonction de la plateforme sur laquelle le logiciel est installé).

La partie *Licensing* est souvent celle qui fait « réagir » lorsque l'on parle de Xamarin. En effet, il faut savoir que les licences sont par plateforme, par développeur et par mois (à moins de payer à l'année). Ainsi, si un développeur veut développer pour iOS et Android, avec Xamarin, il devrait payer (dans le cas d'une licence Indie), 2 fois 25$ par mois (soit

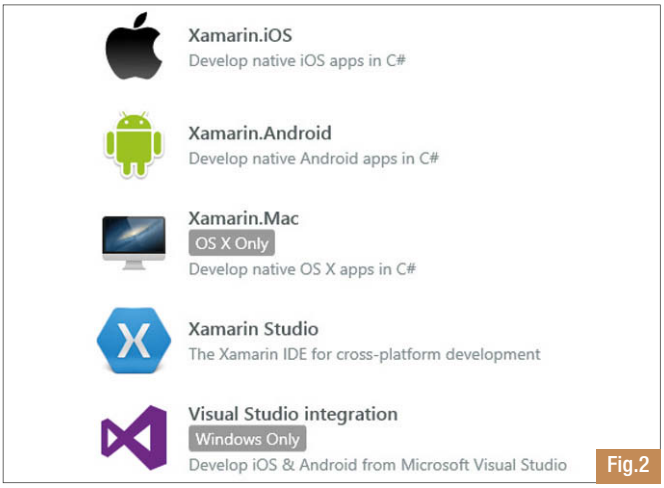


Fig.2

50$ par mois). Dans le cas d'une grosse équipe de développement, le montant accumulé des licences peut donc devenir rapidement conséquent mais ce montant est à mettre en correspondance avec le temps gagné, le fait que les compétences des développeurs .NET ont pu être réutilisées, etc. Vous avez également besoin d'émulateurs qui vous permettent de tester votre application sur différents modèles de périphériques. Pour cela, vous pouvez utiliser les émulateurs installés automatiquement avec l'installation du SDK d'Android (faite automatiquement lors de l'installation de Xamarin) mais ceux-ci sont relativement lents. Vous pouvez donc vous tourner vers d'autres alternatives comme le Xamarin Android Player (téléchargeable ici : https://xamarin.com/android-player), qui offre des performances optimales, le support du glisser/déposer de fichiers APK pour installer des applications, et bien d'autres choses encore (support de la caméra et du clavier, géolocalisation, etc.). Une autre possibilité est d'utiliser GenyMotion (http://www.genymotion.com), un autre émulateur bien connu des développeurs Android. Concernant iOS, le problème est simple : sur votre Mac, il est nécessaire d'installer les outils de développement (XCode, etc.) qui vous permettent d'accéder à l'émulateur iPhone/iPad (de la même façon qu'avec Visual Studio, vous avez accès aux émulateurs Windows Phone).

Démarrer son premier projet Android

Une fois le SDK Xamarin installé, il ne reste plus qu'à commencer le développement de vos premières applications. Si vous lancez Visual Studio et que vous faites « File » -> « New Project », vous pourrez constater que de nouvelles catégories de projets sont disponibles : Fig.4.

Lorsque vous démarrez avec une nouvelle application Android, Visual Studio va créer un projet contenant un ensemble d'éléments :

- "Assets", qui est le répertoire qui va contenir les différents assets utilisables par l'application (une police, etc.),
- "Resources" qui est un répertoire lui-même contenant :
  - "drawable" : qui représente le répertoire contenant des icônes, des images,
  - "layout" : c'est le répertoire qui va contenir les fichiers AXML, utilisés pour représenter les interfaces graphiques des applications Android,
  - "values" : contient les fichiers XML contenant les chaînes de caractères, localisées, utilisées par l'application,
- "MainActivity.cs" : qui représente la première activité Android de l'application (entendre par là qu'il s'agit du premier écran de l'application) Fig.5.

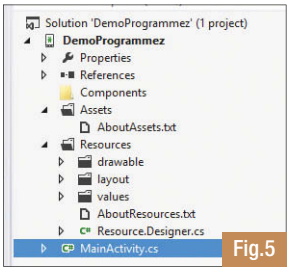


Fig.5

Lorsque l'on commence à développer pour Android, il faut savoir que les APIs natives n'utilisent pas directement des noms de fichiers mais des identifiants. Ainsi, lorsque vous compilez une application qui utilise des ressources, le système va générer une classe Resource qui contient des identifiants pour chacune des ressources utilisées (les « layout », les « drawable », etc.) :

```
namespace DemoProgrammez
{
  public partial class Resource
  {
    public partial class Attribute
    {
      private Attribute()
      {
      }
    }
    public partial class Drawable
    {
      // aapt resource value: 0x7f020000
      public const int Icon = 2130837504;

      private Drawable()
      {
      }
    }

    public partial class Id
    {
      // aapt resource value: 0x7f050000
      public const int MyButton = 2131034112;

      private Id()
      {
      }
    }

    public partial class Layout
    {
      // aapt resource value: 0x7f030000
```

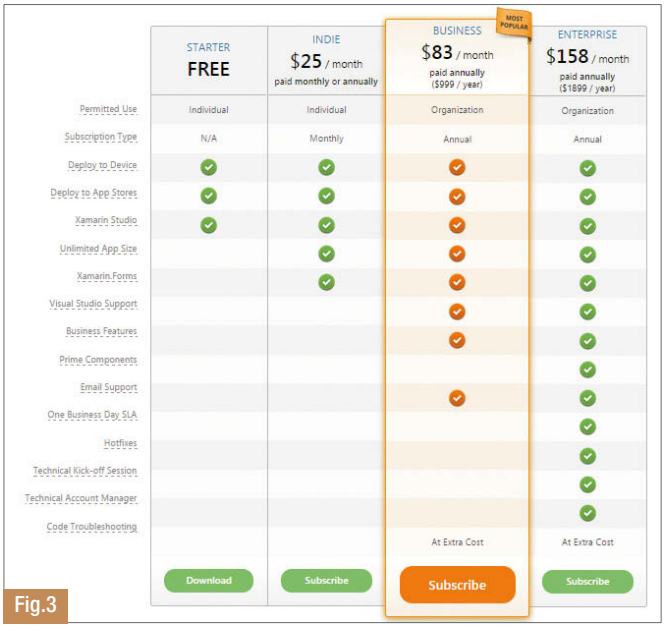


Fig.3

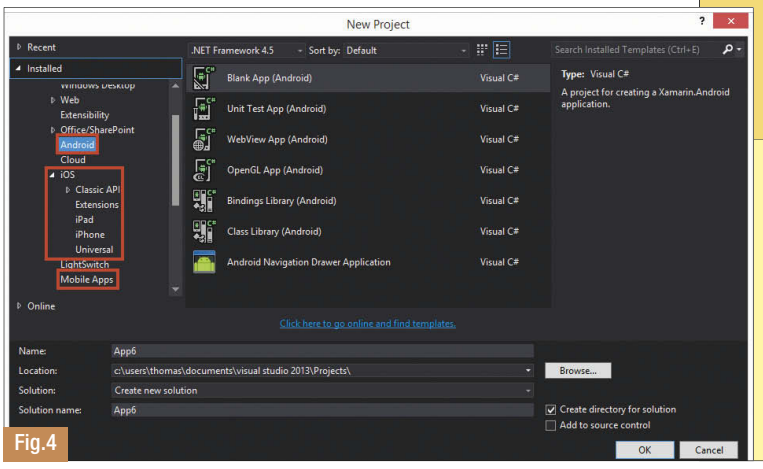


Fig.4

```
    public const int Main = 2130903040;

    private Layout()
    {
    }
  }

  public partial class String
  {
    // aapt resource value: 0x7f040001
    public const int ApplicationName = 2130968577;

    // aapt resource value: 0x7f040000
    public const int Hello = 2130968576;

    private String()
    {
    }
  }
}
```

Dès lors, pour utiliser vos différentes ressources, il vous suffit de passer par la classe correspondante : Fig.6. Au niveau de l'implémentation des activités, tout se passe dans le code de la méthode **OnCreate** :

```
protected override void OnCreate(Bundle bundle)
{
  base.OnCreate(bundle);

  SetContentView(Resource.Layout.Main);
}
```

En utilisant la méthode *SetContentView*, cette méthode OnCreate se charge de charger le fichier contenant l'interface graphique de notre activité (là encore, on remarque l'utilisation de la classe Resource pour accéder à l'identifiant du layout nommé Main).
C'est dans cette méthode OnCreate que l'on va également être en mesure de récupérer une référence sur les contrôles qui auront été définis et positionnés dans le fichier AXML. Pour ce faire, il faut utiliser la méthode FindViewById<T>, méthode générique qui prend en paramètre le type du contrôle auquel on veut accéder ainsi que l'identifiant du contrôle. Ainsi, pour accéder à un bouton nommé MyButton, voici le code nécessaire :

```
Button button = FindViewById<Button>(Resource.Id.MyButton);
```

Les interfaces graphiques, sous Android, sont symbolisées par ces fichiers AXML qui, dans l'absolu, ne sont que des fichiers XML. En double cliquant sur un de ces fichiers, on peut visualiser le designer Android qui permet de glisser/déposer des composants depuis la boîte à outils sur la surface de dessin afin de concevoir le design de l'application : Fig.7. Si vous regardez bien en bas de l'interface, vous pourrez voir un onglet nommé « Source » qui, si vous cliquez dessus, vous permet de visualiser le code XML de la page :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/
apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <Button
    android:id="@+id/MyButton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/Hello" />
</LinearLayout>
```

Bien sûr, vous n'êtes pas obligé de passer par cette vue XML mais il est toujours intéressant de la regarder de temps en temps pour comprendre comment fonctionnent les éléments, quel est leur implication dans la mise en page, etc. Dès lors que la page et l'activité sont implémentées, il ne reste plus qu'à rajouter votre logique afin de gérer la récupération des données, l'affichage, la saisie de données par l'utilisateur, etc.

Démarrer son premier projet iOS

Pour démarrer son projet iOS, la procédure est sensiblement identique. Après avoir choisi de créer un projet iOS, Visual Studio va vous demander de vous associer à un agent de compilation présent sur un Mac, au moyen des fenêtres suivantes : Fig.8 et 9.
Une fois la connexion réussie, Visual Studio vous demande de valider le code correspondant à l'agent de compilation : Fig.10 et 11.
Lorsque vous êtes connecté à l'agent de compilation, Visual Studio a terminé de générer les différents fichiers nécessaires et vous vous retrouvez avec les éléments suivants :

- Le répertoire "Resources" qui contient les différentes ressources utilisables par

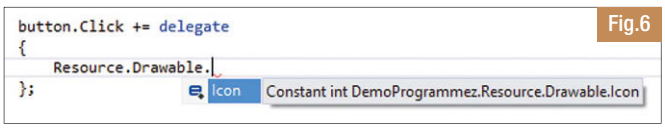


Fig.6

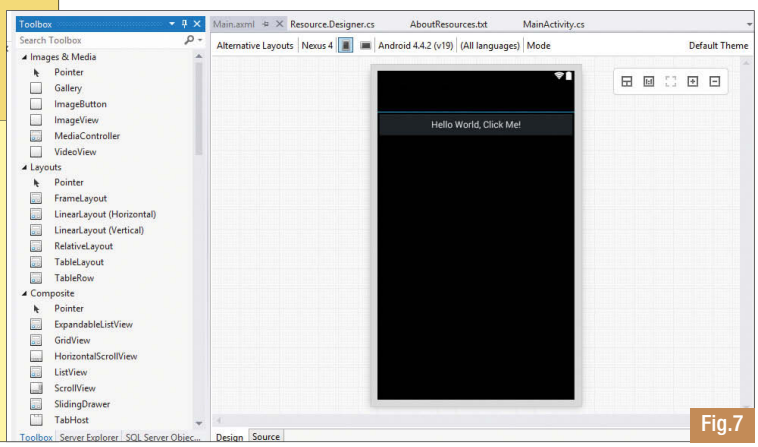


Fig.7

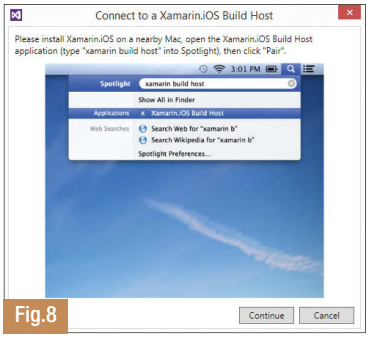


Fig.8

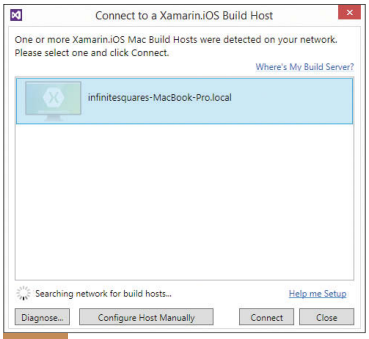


Fig.9

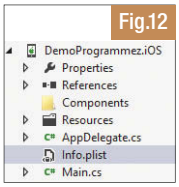


Fig.12

l'application (images, polices, etc.),
- "Mains.cs", qui correspond au point d'entrée de l'application,
- "AppDelegate.cs", qui contient le code permettant de gérer la logique globale de l'application : lancement de l'application, le passage en arrière-plan, etc. Fig.12.

La création d'application iOS passe par la mise en place d'objets héritant de **UIViewController** : Fig.13. Cette classe contient toute la logique associée à la vue (notification de la fin du chargement de la vue, notification lorsque la vue va être supprimée, etc.) :

```
public partial class SampleViewController : UIViewController
{
  static bool UserInterfaceIdiomIsPhone
  {
    get { return UIDevice.CurrentDevice.UserInterfaceIdiom
 == UIUserInterfaceIdiom.Phone; }
  }

  public SampleViewController(IntPtr handle)
    : base(handle)
  {
  }

  public override void DidReceiveMemoryWarning()
  {
    base.DidReceiveMemoryWarning();
  }

  #region View lifecycle

  public override void ViewDidLoad()
  {
    base.ViewDidLoad();
  }

  public override void ViewWillAppear(bool animated)
  {
    base.ViewWillAppear(animated);
  }

  public override void ViewDidAppear(bool animated)
  {
    base.ViewDidAppear(animated);
  }

  public override void ViewWillDisappear(bool animated)
  {
    base.ViewWillDisappear(animated);
  }

  public override void ViewDidDisappear(bool animated)
  {
    base.ViewDidDisappear(animated);
  }

  #endregion

}
```

La création d'interfaces graphiques repose sur la création de fichiers portant les extensions « xib » ou « storyboard ». Grâce à la connexion au Mac, il est même possible d'avoir un designer intégré directement dans Visual Studio ! Fig.14. Dans les faits, ces fichiers ne sont que des fichiers XML qui disposent d'un éditeur permettant de les interpréter et de les présenter de manière plus agréable pour l'utilisateur :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<document  type="com.apple.InterfaceBuilder3.CocoaTouch.
Storyboard.XIB" version="3.0" toolsVersion="4451" system
Version="13A461" targetRuntime="iOS.CocoaTouch" property
AccessControl="none" useAutolayout="YES" initialViewController
="vXZ-lx-hvc">
  <dependencies>
    <plugIn identifier="com.apple.InterfaceBuilder.IBCocoa
TouchPlugin" version="3733.0"/>
  </dependencies>
  <scenes>
    <!--class Prefix:identifier View Controller-->
    <scene sceneID="ufC-wZ-h7g">
      <objects>
        <viewController id="vXZ-lx-hvc" customClass="Sample
```

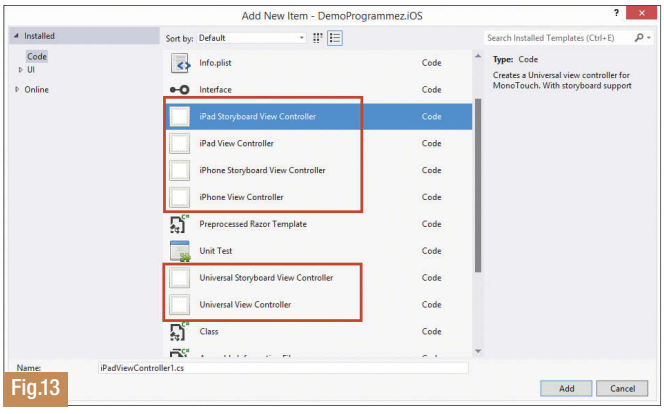


Fig.13

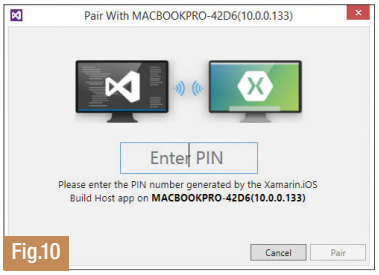


Fig.10

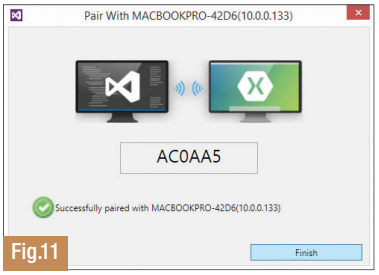


Fig.11

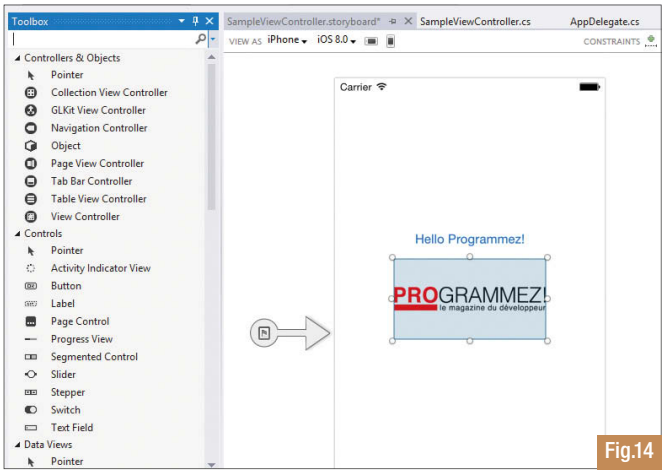


Fig.14

```
ViewController" sceneMemberID="viewController" storyboard
Identifier="SampleViewController"  restorationIdentifier=
"SampleViewController">
          <layoutGuides>
            <viewControllerLayoutGuide type="top" id="3"/>
            <viewControllerLayoutGuide type="bottom" id="4"/>
          </layoutGuides>
          <view key="view" contentMode="scaleToFill" id="kh9
-bI-dsS">
            <rect key="frame" x="0.0" y="0.0" width="320"
 height="568"/>
            <autoresizingMask key="autoresizingMask" flexible
MaxX="YES" flexibleMaxY="YES"/>
            <color key="backgroundColor" white="1" alpha="1"
 colorSpace="custom" customColorSpace="calibratedWhite"/>
            <subviews>
              <button opaque="NO" contentMode="scaleToFill"
 contentHorizontalAlignment="center" contentVerticalAlignment
="center"  buttonType="roundedRect"  lineBreakMode="middle
Truncation" id="5" fixedFrame="YES" translatesAutoresizing
MaskIntoConstraints="NO">
                <rect key="frame" x="88" y="150" width="145"
 height="80"/>
                <state key="normal" title="Hello Programmez!">
                  <color key="titleShadowColor" white="0.5"
 alpha="1" colorSpace="calibratedWhite"/>
                </state>
                <connections>
                  <action selector="UIButton5_TouchUpInside:
" destination="vXZ-lx-hvc" id="6" eventType="touchUpInside"/>
                </connections>
              </button>
              <imageView userInteractionEnabled="NO" content
Mode="scaleAspectFit"  id="7"  fixedFrame="YES"  translates
AutoresizingMaskIntoConstraints="NO" image="logo.png">
                <rect key="frame" x="70" y="214" width="180"
 height="95"/>
              </imageView>
            </subviews>
          </view>
        </viewController>
        <placeholder placeholderIdentifier="IBFirstResponder"
 id="x5A-6p-PRh" sceneMemberID="firstResponder"/>
      </objects>
    </scene>
  </scenes>
  <simulatedMetricsContainer key="defaultSimulatedMetrics">
    <simulatedStatusBarMetrics key="statusBar"/>
    <simulatedOrientationMetrics key="orientation"/>
    <simulatedScreenMetrics key="destination" type="retina4"/>
  </simulatedMetricsContainer>
  <resources>
    <image name="Default-568h.png" width="640" height="1136"/>
    <image name="logo.png" width="512" height="85"/>
  </resources>
</document>
```

C'est dans la classe héritant de UIViewController que l'on va retrouver toute la logique associée à l'interface graphique. C'est aussi là que l'on va se charger de se connecter à un Web Service, de récupérer les données, etc. Ainsi, dans l'exemple précédent, si on double-clique sur le bouton, on se retrouve automatiquement dans le fichier de code associé, et c'est là que nous pouvons rajouter notre code :

```
partial void UIButton5_TouchUpInside(UIButton sender)
{
  var alertView = new UIAlertView("Hello", "Bienvenue dans
 cette application!", null, "Annuler");
  alertView.Show();
}
```

Lors de l'exécution, on choisit la cible d'exécution de notre application (un iPhone physique ou le simulateur) et un simple appui sur F5 permet de tester le résultat : Fig.15 et 16.
Attention, pour que cela fonctionne, il est nécessaire de modifier la séquence d'initialisation de l'application pour indiquer que l'on souhaite démarrer par le Storyboard que l'on vient de créer. Pour cela, il faut modifier le code du fichier AppDelegate.cs comme suit :

```
[Register("AppDelegate")]
public partial class AppDelegate : UIApplicationDelegate
{
  UIWindow window;

  public UIStoryboard sampleViewController = UIStoryboard.
FromName("SampleViewController", null);
  public UIViewController initialViewController;

  public override bool FinishedLaunching(UIApplication app,
 NSDictionary options)
  {
    window = new UIWindow(UIScreen.MainScreen.Bounds);

    initialViewController = sampleViewController.Instantiate
InitialViewController() as UIViewController;

    window.RootViewController = initialViewController;

    window.MakeKeyAndVisible();

    return true;
  }
}
```

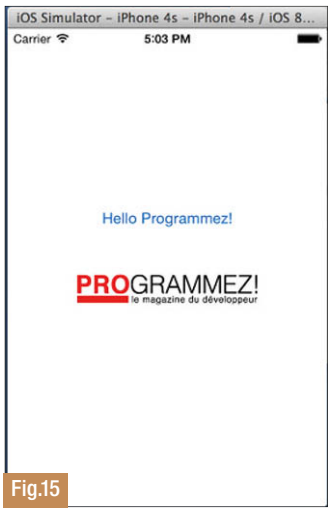


Fig.15

Fig.16

Comme vous pouvez le constater, l'implémentation est relativement simple et repose sur le fait d'instancier un Storyboard par son nom, puis de créer un objet de type ViewController à partir du Storyboard !

Comment déployer son binaire sur les Stores ?

Votre application est complètement développée, vous l'avez testée sur différents périphériques et vous êtes maintenant prêt à la déposer sur les Stores de Google et d'Apple.

Pour Android, la procédure est simple car elle consiste à générer une clé de cryptage (un certificat) et à utiliser cette clé pour créer le fichier APK qui pourra ensuite être déposé sur le Store. Pour cela, il faut commencer par changer la configuration de compilation, dans Visual Studio, en se positionnant sur la configuration « Release » : Fig.17.

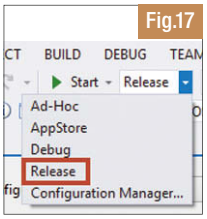


Fig.17

Après avoir compilé le projet, il faut cliquer sur le menu « **Tools** » puis « **Android** » et enfin « **Publish Android App** » : Fig.18.

Là, Visual Studio vous demande si vous souhaitez créer une nouvelle clé ou, si vous aviez déjà fait cette étape, si vous désirez utiliser une clé existante : Fig.19, 20 et 21.

Saisissez les différentes informations demandées puis cliquez sur le bouton « *Create* ». Cela aura pour effet de déclencher la chaîne de compilation et de signature (il est possible que vous aperceviez différentes fenêtres de commandes MS-DOS qui s'ouvrent puis se ferment instantanément) et, une fois terminé, vous pouvez vous rendre dans le répertoire que vous avez indiqué pour trouver un fichier APK suffixé par « -Aligned » : c'est ce fichier que vous allez pouvoir déposer sur le Store de Google pour soumettre votre application ! Fig.22.

Côté iOS, le processus de publication est un peu différent et, surtout, un peu plus long. En effet, il est nécessaire de configurer la création du profil de « provisioning » directement depuis le portail d'Apple (accessible à l'adresse : http://developer.apple.com).

Une fois le profil créé, il va être possible de générer le certificat associé en utilisant l'outil « *Keychain Access* » (sur le Mac) : Fig.23.

Ensuite, il faut aller dans les propriétés du projet puis, dans l'onglet « *iOS Bundle Signing* », sélectionner la configuration « *AppStore* » et la plateforme « *iPhone* » (qui permet de représenter un périphérique physique). Dans la partie « *Identity* », sélectionner « *Distribution* (*Automatic*) » et, dans le « *Provisioning profile* », sélectionner le profil que vous avez créé précédemment puis lancer une compilation de votre application : Fig.24.

Une fois la compilation terminée, vous pouvez envoyer votre application sur le Store d'Apple, en vous rendant à l'adresse suivante : https://itunesconnect.apple.com. Pour en savoir plus sur la publication d'application iOS, je vous recommande ce lien qui explique, de manière très détaillée, le processus de publication : http://developer.xamarin.com/guides/ios/deployment,_testing,_and_metrics/app_distribution_overview/publishing_to_the_app_store/.

Le mot de la fin

Xamarin est une technologie en pleine expansion, qui offre de nombreuses possibilités lorsque l'on parle de développement cross-plateformes. Certes, il y a encore des choses à travailler pour arriver à quelque chose d'optimal et des axes d'amélioration/travail existent mais, en l'état actuel, il serait dommage de ne pas s'y intéresser si vous avez des besoins d'applications iOS/Android et une équipe disposant de compétences .NET/C# ! ⊞

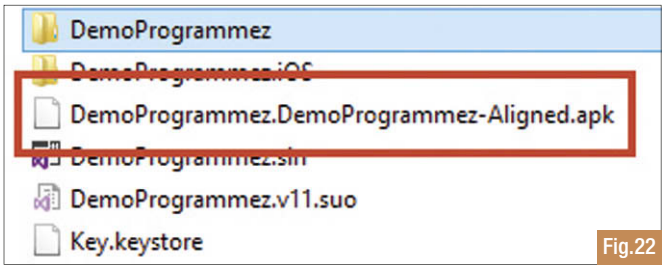


Fig.22

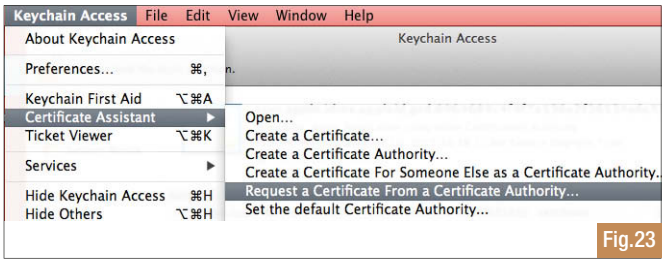


Fig.23

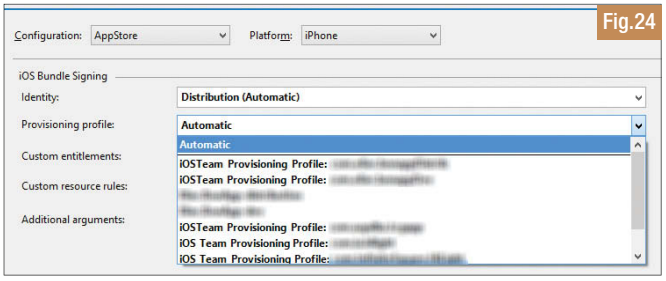


Fig.24

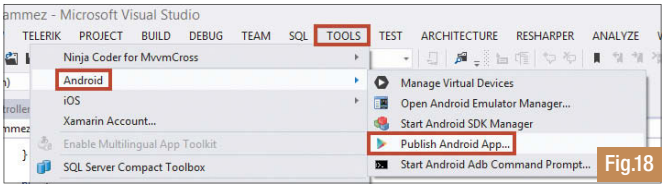


Fig.18

Fig.19

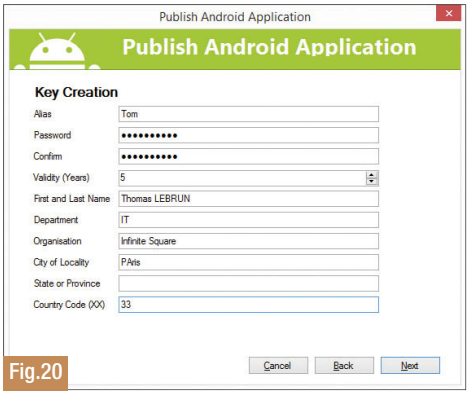


Fig.20

Fig.21

Utilisation du pattern MVVM avec Xamarin

Le pattern MVVM (Model View ViewModel) est devenu le pattern de référence des développeurs XAML. Dès lors, il devient logique, pour ces développeurs, d'essayer de retrouver leur marque et d'appliquer ce pattern lorsqu'ils doivent développer pour Android et iOS avec Xamarin.

Thomas LEBRUN – *Consultant*
Infinite Square
http://blogs.infinitesquare.com/b/tom
@thomas_lebrun

Des Frameworks MVVM, il en existe beaucoup mais peu sont également compatibles avec Android et iOS, comme c'est le cas de MvvmCross et MvvmLight Toolkit, que nous allons détailler.

Présentation de MvvmCross

MvvmCross est un Framework MVVM développé par Stuart Lodge (http://slodge.blogspot.fr) et accessible sur GitHub à l'adresse suivante : https://github.com/MvvmCross/MvvmCross

Comme beaucoup, il propose des fonctionnalités « classiques » comme la mise à disposition d'un ViewModel de base, de classe implémentant l'interface ICommand (pour gérer les différentes commandes), un container IoC rapide et efficace, etc. Mais ce qu'il a de bien (ou, a minima, de plus que les autres), c'est qu'il dispose d'un mécanisme par attribut permettant, lorsque l'on développe son application Android, de retrouver une syntaxe proche de ce que l'on connait en XAML. Ainsi, voici la structure classique d'un projet avec MvvmCross : Fig.1. Des fichiers supplémentaires sont présents dans le cas ci-dessus mais, en essence, on dispose de 2 projets :

- Un projet « Core », qui va contenir les ViewModels, les utilitaires ainsi que tous les éléments partagés
- Un projet Android qui va contenir les différentes activités (entendre par là les vues) accessibles à l'utilisateur

A noter qu'il est possible d'avoir 3 projets dans la solution, dans le cas où un projet iOS est également présent. Au sein du projet « *Core* », on retrouve une classe App.cs, qui symbolise le point de démarrage de l'application : c'est là que l'on va pouvoir faire de l'initialisation de données, enregistrer des services dans le container IoC, etc Fig.2. Cette classe App.cs est ensuite utilisée dans chacun des projets Android ou iOS, à travers la classe de mise en place de l'application, gérée dans le fichier Setup.cs : Fig.3. L'idée de MvvmCross étant de vous permettre d'utiliser le pattern MVVM au sein de vos applications iOS et Android, on dispose donc de méthodes et utilitaires permettant de nous faciliter le travail. Ainsi, dans le cas d'Android, l'attribut MvxBind est disponible, prenant en paramètres :

- La propriété graphique du contrôle que l'on veut binder
- La propriété CLR de l'objet utilisé comme source du binding (bien souvent, il va s'agir d'une propriété de votre ViewModel) Fig.4.

MvvmCross supporte également l'utilisation des convertisseurs de valeur, qui permettent de prendre un objet d'un certain type et de renvoyer un objet d'un autre type (en XAML, ce mécanisme est implémenté via l'interface *IValueConverter*). Cependant, plutôt que de travailler avec une interface dont les méthodes prennent en paramètres des objets de type '*object*', MvvmCross simplifie le processus en mettant à disposition une classe *MvxValueConverter<TFrom, TTo>* : Fig.5. Pour utiliser le convertisseur nouvellement créé, il suffit de le référencer par son nom : Fig.6. Pour identifier que cette classe est un convertisseur utilisable, MvvmCross se base sur l'utilisation de la réflexion et la convention de nommage (les convertisseurs ayant leur nom se terminant par « Converter ») mais ce mécanisme est entièrement modifiable si vous le souhaitez ! L'utilisation des commandes est également possible avec MvvmCross. Là encore, il faut passer par une classe dédiée, MvxCommand, qui implémente l'interface ICommand. Mais si vous êtes habitué au RelayCommand, vous ne serez pas dépaysé : Fig.7. Bien sûr, toutes ces fonctionnalités sont également accessibles sous iOS, en utilisant la syntaxe dite « Fluent », qui propose un grand nombre de possibilités comme cela est visible dans Visual Studio : Fig.8. Attention, dans ce cas, il ne faut pas oublier d'appliquer le binding en utilisant la méthode **Apply** :

Fig.1

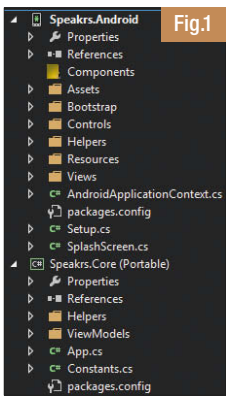


Fig.4

```
<LinearLayout
    android:orientation="horizontal"
    android:minWidth="25px"
    android:minHeight="25px"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/linearLayout1"
    android:layout_gravity="center_horizontal">
    <TextView
        local:MvxBind="Text CurrentSpeaker.FirstName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/detailsSpeakerFirstName"
        android:textSize="25dp" />
    <TextView
        local:MvxBind="Text CurrentSpeaker.LastName"
        android:textAllCaps="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/detailsSpeakerLastName"
        android:layout_marginLeft="5dp"
        android:textSize="25dp" />
</LinearLayout>
```

Fig.3

```
public class Setup : MvxAndroidSetup
{
    0 references
    public Setup(Context applicationContext) : base(applicationContext)
    {
    }

    1 reference
    protected override void InitializeIoC()
    {
        base.InitializeIoC();

        Mvx.RegisterSingleton(typeof(IApplicationContext), () => new AndroidApplicationContext());
    }

    0 references
    protected override IMvxApplication CreateApp()
    {
        return new Core.App();
    }

#if DEBUG

    0 references
    protected override IMvxTrace CreateDebugTrace()
    {
        return new DebugTrace();
    }

#endif
}
```

Fig.6

```
<MvxImageView
    local:MvxBind="ImageUrl Picture, Converter=StringToImageConverter, ConverterParameter=HelloWorld!"
    android:src="@android:drawable/ic_menu_gallery"
```

Fig.2

```
1 reference
public class App : Cirrious.MvvmCross.ViewModels.MvxApplication
{
    0 references
    public override void Initialize()
    {
        QueryBase.BaseUrl = Constants.BASE_URL;

        typeof(QueryBase).GetTypeInfo().Assembly
            .CreatableTypes()
            .EndingWith("Query")
            .AsTypes()
            .RegisterAsLazySingleton();

        RegisterAppStart<MainViewModel>();
    }
}
```

Fig.5

```
0 references
public class StringToImageConverter : MvxValueConverter<string, byte[]>
{
    0 references
    protected override byte[] Convert(string value, Type targetType, object parameter, CultureInfo culture)
    {
        return base.Convert(value, targetType, parameter, culture);
    }

    0 references
    protected override string ConvertBack(byte[] value, Type targetType, object parameter, CultureInfo culture)
    {
        return base.ConvertBack(value, targetType, parameter, culture);
    }
}
```

```
var bindingSet = this.CreateBindingSet<BaseView, DemoViewModel>();
bindingSet.Bind(label)
  .For(l => l.Text)
  .To(vm => vm.LastName);

bindingSet.Apply();
```

A noter : Si Visual Studio ne reconnaît pas la méthode CreateBindingSet, rajouter le using suivant dans le fichier de code source :

```
using Cirrious.MvvmCross.Binding.BindingContext;
```

Cela aura pour effet d'importer les méthodes d'extension disponibles avec MvvmCross.

MvvmLight Toolkit pour Xamarin

MvvmLight Toolkit est un Framework MVVM, extrêmement populaire, développé par Laurent Bugnion (http://blog.galasoft.ch), accessible sur son site Codeplex : http://mvvmlight.codeplex.com.

Il a été initialement développé pour les technologies XAML avec un objectif simple : mettre à disposition des développeurs les fonctionnalités les plus simples et élémentaires lorsque l'on veut implémenter le pattern MVVM et c'est tout. Ainsi, on ne retrouve pas de mécanisme de plugins, de fonctionnalités avancées, etc. Sous l'influence de la popularité de Xamarin, Laurent a implémenté une version du Toolkit pour Xamarin et nous allons voir comment l'utiliser dans nos applications.

Pour utiliser ce Framework MVVM, il faut commencer par rajouter le packages Nuget contenant les librairies de MVVM Light.

Ensuite, le développement est assez similaire à ce que l'on a l'habitude de faire. Ainsi, on commence par créer son ViewModel :

```
public class MainViewModel : ViewModelBase
{
  private string _lastName;

  public string LastName
  {
    get
    {
      return this._lastName;
    }
    set
    {
      this.Set(() => this.LastName, ref this._lastName, value);
    }
  }

  public RelayCommand SelectCommand { get; set; }

  public MainViewModel()
  {
    this.SelectCommand = new RelayCommand(() =>
    {
      // Work with LastName
    },() => !string.IsNullOrWhiteSpace(LastName));
  }
}
```

Celui-ci hérite de ViewModelBase, une classe mise à disposition par MVVM Light Toolkit, qui offre des fonctionnalités telles que la communication entre ViewModels (avec le système de Messenger), la notification de l'interface graphique (avec le déclenchement de l'évènement PropertyChanged), etc. Ensuite, il faut utiliser ce ViewModel dans l'activité Android (ou le ViewPresenter sur iOS), en utilisant les méthodes d'extension spécialement créées pour cela, à savoir :

- SetBinding, pour lier la propriété graphique d'un contrôle à la propriété CLR de notre ViewModel
- SetCommand, pour lier une commande à un évènement d'un contrôle

Le code de notre activité devient donc le suivant :

```
private MainViewModel _mainViewModel;
public MainViewModel MainViewModel
{
  get { return _mainViewModel ??
    (_mainViewModel = new MainViewModel()); }
}
protected override void OnCreate(Bundle bundle)
{
  base.OnCreate(bundle);

  SetContentView(Resource.Layout.Main);

  Button button = FindViewById<Button>(Resource.Id.MyButton);

  this.SetBinding(
    () => MainViewModel.LastName,
    () => button.Text);

  this.SetCommand("Click", MainViewModel.SelectCommand);
}
```

On constate que la mise en place, au-delà d'être très rapide, est surtout très simple et accessible à n'importe quel développeur susceptible de travailler sur votre application.

En conclusion

MvvmCross ou MVVM Light Toolkit, le choix n'est pas simple: il faut bien garder en tête que chacun dispose de ses propres fonctionnalités, avec ses propres objectifs. Si vous êtes amené à faire ce choix, je ne saurais que vous conseiller de bien étudier la question : quels sont vos besoins ? Que souhaitez-vous faire ? Car, dans l'absolu, il n'est pas impossible que ni l'un ni l'autre ne corresponde à vos besoins et que vous soyez obligé de regarder ailleurs.

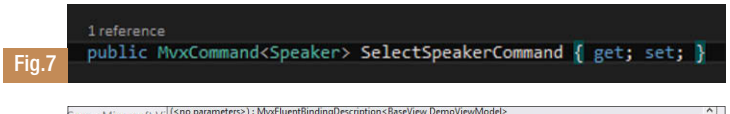


Fig.7

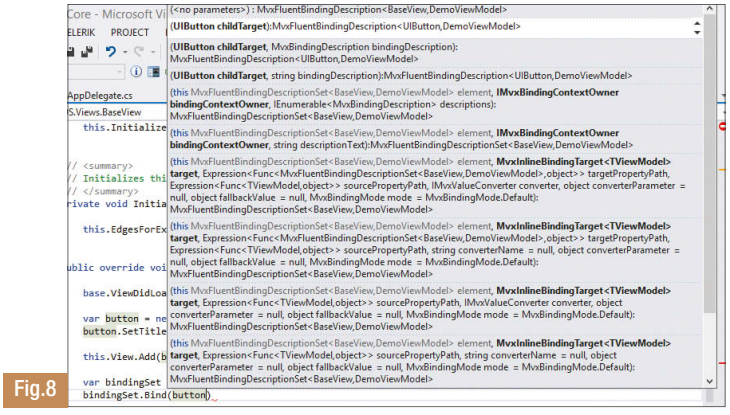


Fig.8

Penser parallèle, penser multithread

En automne 2010, nous avons réalisé pour Intel un livre blanc intitulé « le multicore, la programmation parallèle et le développeur : pourquoi penser parallèle ». Dès l'introduction, nous évoquions une réalité : « Le parallélisme est une autre manière de penser son code, son architecture logicielle, son fonctionnement. Bref, il faut : **penser parallèle. Coder parallèle.** *». Nous nous demandions s'il ne fallait pas créer un langage parallèle dédié, P# ou le langage P.*

Il y a 5 ans, le parallélisme, la programmation multi-cœur, était un peu une mode. Des éditeurs comme Intel, Microsoft, et même Apple, communiquaient beaucoup sur ces techniques. Mais ces techniques, pour la plupart, restaient relativement difficiles à maîtriser. Des librairies comme PLinq (Microsoft / .Net) permettent de paralléliser des fonctions et boucles précises en réduisant à quasi rien les modifications de codes. Aujourd'hui, de nombreux logiciels, notamment les plus exigeants en ressources, utilisent plusieurs cœurs sur nos machines qui en possèdent souvent 4 ou 8, voire, plus pour certaines configurations. Les terminaux mobiles possèdent eux aussi plusieurs cœurs.

Le parallélisme n'apporte pas uniquement des gains de performances. Il permet de découper le fonctionnement des applications en tâches et threads, de ne pas bloquer le fonctionnement, de proposer des applications toujours réactives.

S'il existe des librairies, des frameworks pour faciliter la parallélisation du code, c'est à dire l'exécution du code (donc de l'application) sur plusieurs cœurs simultanément, ce travail de haut niveau ne permet pas une finesse d'optimisation et l'architecture de l'application reste peu ou prou la même. Paralléliser (quelle que soit la méthode utilisée) un code existant n'a rien de trivial.

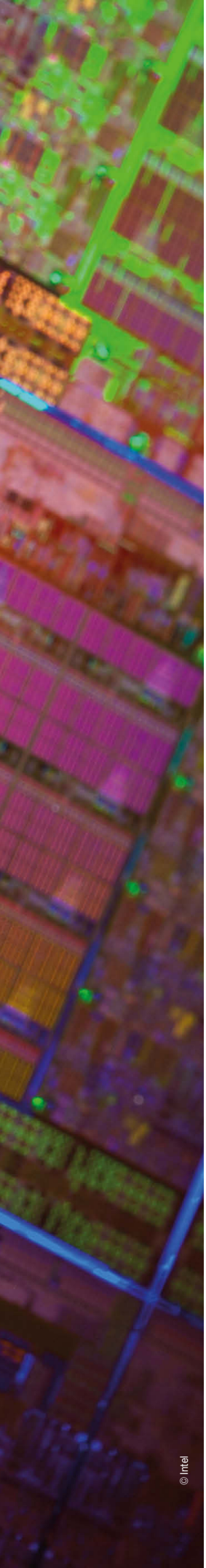
Le terme parallélisme recouvre une grande variété d'éléments : multithreading, multitâche, multi / many-cœur, parallélisme des données et des traitements, asynchronisme, GPGPU, etc. Car le parallélisme peut concerner aussi bien l'application que les données.

Pourquoi paralléliser du code ? Il existe de multiples usages et scénarii :

- *Compression / décompression,*
- *Exécution parallèle de tâches et de flux complexes et lourds,*
- *Traitement audio / vidéo / imagerie,*
- *Amélioration du comportement d'une application et éviction des blocages,*
- *Améliorer les performances et assurer une montée en charge de l'application,*
- *Transformation 2D / 3D, contenu relief, 3D augmentée, technologies vocales, les jeux, la cryptographie, etc.*
- *HPC,*
- *Accélération des moteurs de rendu HTML, javascript...*
- *Utiliser les ressources processeurs.*

Nous vous proposons de revenir sur quelques notions et outils du parallélisme et du HPC. Nous verrons comment on débugge du code parallèle ou encore comment on utilise la librairie OpenMP. OpenMP est une des librairies les plus utilisées dans le calcul parallèle intensif.

La rédaction


© Intel

Projet Exascale : 10 cœurs ? Non, 100 millions de cœurs processeurs physiques !

Exascale peut se définir comme le futur des calculs intensifs et du massivement parallèle. L'objectif est à la fois simple et d'une extrême complexité : concevoir des algorithmes, des librairies, des outils capables de fonctionner et d'exploiter 100, voire, 120 millions de cœurs pour puissance de calculs monstrueuse en exaflops, soit 10 puissance 18 ! Les premières architectures devraient voir le jour dès 2018 – 2019 !

Aller où le processeur n'est jamais allé

Exascale doit permettre de dépasser les limites techniques et technologiques actuelles. La loi de Moore n'est plus tenable. La gravure processeur est passée de 90nm à 14 nm. On parle déjà d'une gravure à 7 nm ! Mais d'énormes problèmes doivent être surmontés car les lois physiques induisent une instabilité et des fuites d'électrons avec des gravures trop fines. La densité au cm2 pose elle aussi des problèmes. Au niveau architecture machine, nous atteignons des limites. Aujourd'hui, les plus gros calculateurs dépassent 80 000 processeurs (3 millions de cœurs physiques), pour une puissance de 33 petaflops annoncés et utilisables pour une consommation énergétique gargantuesque de 17,8 MW !

Déjà à ce niveau, les outils, les langages, les libraires mathématiques et d'optimisation du code atteignent leurs limites dans la stabilité et la montée en charge. Car répartir la charge entre tous les cœurs et exploiter chaque processeur requiert des développeurs et des ingénieurs de très haut niveau.

Le laboratoire Exascale installé près de Paris au campus Terratec, avec une vingtaine de chercheurs, ingénieurs et développeurs, est là pour observer et comprendre les limites des outils actuels et proposer des solutions pour dépasser ces contraintes et passer du calcul "petaflopique" à la puissance "exaflopique".

Késako le projet ?

Le laboratoire français fait partie d'un réseau de recherche européen (Intel EMAE HPC Exascale labs, Intel étant très actif sur ces recherches) et il fait parti du EXA2CT. Le laboratoire (Exascale Computing Research) fut fondé en 2010 par le CEA, Genci, Intel et l'UVSQ. Le lab a créé, avec des contributions universitaires, des spécifiques : MAQAO. Ces outils, en open source permettent d'analyser les performances des applications HPC. Il supporte aussi les processeurs Xeon Phi (Phi comporte plusieurs dizaines de cœurs). Autre contribution : Codelet Tuning Infrastructure (CTI), disponible aussi en open source. Cet outil de « CTI permet de constituer un répertoire de codelets -quelques lignes de code représentant les parties les plus critiques- extraites de l'application complète avec leur environnement runtime, qui sont ré-exécutées en vue de procéder à l'exploration d'options de compilation ou de différentes architectures. », dixit la présentation officielle.

Comme vous l'aurez compris : Exascale doit étudier et résoudre les problématiques autour des algorithmes, les architectures HPC, les modèles de développement. Ce vaste projet doit créer des prototypes d'applications et d'outils, en open source.

D'autres projets Exascale européens existent. Par exemple, le projet Mont-Blanc se découpe en deux phases :

- 2011-2015 : développer des prototypes matériels de machines HPC énergiquement plus efficaces, utilisant des composants et technologies venant des fabricants de systèmes embarqués, créer des applications et outils Exascale,
- 2013-2016 : se concentrer sur la partie logicielle du projet (système, applications, outils de développement). Support des processeurs ARM 64.

Des constructeurs comme Bull investissent aussi sur la recherche Exascale avec pour ambition de construire un supercalculateur exa (SEQUANA), de créer une pile logicielle adaptée (bullx supercomputer suite). Pour Bull, l'approche exa fait parti de l'Extreme Computing.

Quels débouchés sur le HPC ?

Il s'agit d'un marché très spécifique. Même si le HPC est aujourd'hui un outil crucial pour de nombreuses entreprises (et pas seulement pour les grandes entreprises), particulièrement la simulation. En termes d'emploi, un développeur ne devient pas expert HPC comme cela. Comme nous l'a précisé Guillaume Colin de Verdière du CEA, ce sont surtout des ingénieurs et universitaires qui utilisent le HPC. Nous sommes tout de même dans un univers scientifique. Il faut être à l'aise avec le numérique, les mathématiques.
Les développeurs jouent un rôle important en HPC pour concevoir les outils. Un des langages populaires est le Fortran. L'expérience des infrastructures et programmations massivement parallèles est primordiale. Le marché de l'emploi y est faible.

TotalView et ReplayEngine : comment trouver un bug non déterministe dans une application parallèle ?

Le débogage est une activité qui fait partie intégrante du cycle de développement d'une application : si vous développez, alors nécessairement, vous déboguez (penser et écrire du code pour qu'il effectue correctement une action précise, c'est déjà déboguer !). La recherche de bogues à proprement parler est, quant à elle, une activité qui est rarement considérée comme « valorisante ». Plutôt assimilée à une contrainte et à une perte de temps, elle se révèle parfois fastidieuse mais surtout indispensable.

Sébastien Grimonet
Rogue Wave

Le parallèle n'est pas le séquentiel

Dans le cas d'une application séquentielle, on peut raisonnablement estimer que notre façon de penser (séquentielle) va nous permettre de converger avec assez peu d'effort vers la source du problème : en suivant méticuleusement la progression de notre unique processus, nous devrions pouvoir déterminer ce qui pose problème, ce n'est finalement qu'une question de temps et éventuellement d'outillage.

En revanche, dans le cadre d'une application (ou d'un code) parallèle, cela devient nettement différent. Déjà, la notion de conception d'une application où plusieurs processus (tâches) fonctionneraient en parallèle n'est pas une notion tout à fait naturelle à appréhender. Dans un tel contexte, on se retrouve très vite confrontés à des questions légitimes mais embarrassantes du type : « comment découper les tâches en sous-tâches ? », « comment et à quel processeur/cœur attribuer ces sous-tâches ? », « quid de la synchronisation entre ces tâches ? », « quid de la gestion de la mémoire ? » etc. Bien sûr, il existe des méthodes et des standards (citons notamment MPI et OpenMP, très connus dans le milieu du calcul scientifique) sur lesquels s'appuyer pour bâtir de telles applications; avec de l'expérience, on arrive à identifier rapidement les algorithmes qui se prêtent bien à un exercice de parallélisation. Pour ce qui est du débogage de telles applications, il faut là-aussi prendre en compte cette nouvelle couche de complexité : alors que nous sommes « programmés » de manière séquentielle, comment allons-nous pouvoir procéder avec une application parallèle ? D'autant plus que le parallélisme expose les développeurs à de nouveaux types de bogues, comme les *deadlocks* (le programme ne se termine jamais, car, par exemple, 2 processus s'attendent mutuellement), ou les *race conditions* (mauvais ordonnancement des processus, conduisant à des situations inattendues et difficilement reproductibles). Il faut ajouter à cela l'échelle du problème en termes de nombre de processus/*threads* qui s'exécutent en simultané : tenter de déboguer 2 processus concurrents comme on le ferait pour 1 seul reste envisageable, mais si on envisage de « vrais » code HPC pouvant impliquer 4000 processus MPI exécutant chacun 6 *threads* OpenMP ! Ce n'est certes pas la majorité des cas – et il est parfois possible de reproduire le bug à une échelle plus modeste – mais il faut bien noter que les besoins en calcul parallèle sont croissants et que les méthodes et les outils de débogage doivent s'y adapter.

L'application MPI_Replay_Engine_demo.c (code sur programmez.com) est une implémentation parallèle MPI en C++ d'un algorithme de tri fusion (http://fr.wikipedia.org/wiki/Tri_fusion) sur un tableau de réels (1000 éléments dans notre cas). Le principe consiste d'abord à trier des sous-ensembles de ce tableau avant de les réassembler par comparaison d'éléments et fusion. On profitera également de ce partitionnement pour calculer les valeurs maximales locales (`local_max`) et en déduire la valeur maximale globale (`global_max`) du tableau. Les éléments du tableau seront classés par ordre croissant.

```
> mpicxx -O0 -g -DUSEMPI -DMPICH_IGNORE_CXX_SEEK ./MPI_Replay
_Engine_demo.C -o MPI_Replay_Engine_demo
> mpiexec.hydra -n 6 ./MPI_Replay_Engine_demo
```

Si vous exécutez plusieurs fois l'application de manière consécutive, vous allez vous rendre compte que dans certains cas (mais pas systématiquement), la valeur maximale affichée est fausse car elle est inférieure à au moins une des valeurs de la série triée (il suffit souvent de regarder parmi les 10 dernières valeurs affichées).

Déterministe vs non-déterministe

Nous sommes donc confrontés à un bug non-déterministe. C'est-à-dire que même en examinant attentivement le déroulement de notre application, que ça soit au travers d'un débogueur ou en utilisant des `print` un peu partout, rien ne garantit que l'erreur ne se manifeste... De plus, nous nous retrouvons dans le cas d'une application parallèle, pour laquelle chaque process effectue les mêmes opérations que ses confrères. Est-ce que tous les process contribuent de manière identique à ce bug ? Doit-on se concentrer sur un process en particulier ? Si oui, lequel ? Vous avez compris que ce cas de figure est particulièrement difficile à traiter. Aussi, nous allons voir comment il est possible de déterminer la cause du problème à l'aide du débogueur parallèle TotalView et de ses fonctionnalités de débogage inverse.

Pour démarrer TotalView sur notre application parallèle :

```
> totalview -args mpiexec.hydra -n 6 ./MPI_Replay_Engine_demo
```

[Go -> Continue -> Focus sur le process de rang 0 et main dans la Stack Trace]

Nous allons d'abord examiner la répartition des valeurs contenues dans `full_domain` avant qu'il ne soit trié `full_domain`. Pour ce faire :

1 - On place un breakpoint devant l'instruction `MPI_Bcast(…)` par exemple [L60],
2 - On presse le bouton Go. Tous les process vont progresser jusqu'à ce breakpoint,

Suite page 51

RETOURS D'EXPÉRIENCES TECHNIQUES

DES STARTUPS VOUS PRÉSENTENT LEUR UTILISATION DE L'OPEN SOURCE DANS AZURE

salezeo
SALES PROSPECTING REINVENTED

Captain Contrat

ALKEMICS

MoviDone

semsoft

Microsoft

Microsoft Azure

édito

Le Cloud, composant incontournable du développeur

Le Cloud Computing a réellement émergé en 2008. Vu comme une bête curieuse, le développeur a testé, regardé les différents services. Le SaaS a été la première couche à rencontrer le succès. Si l'entreprise a mis plusieurs années à y venir (avec le IaaS et le PaaS), la startup s'est posée moins de questions. Le Cloud apportait une flexibilité et une liberté que l'on ne pouvait pas avoir avec une infrastructure classique ou en infogérance.

En limitant les investissements, la Startup peut monter des prototypes, tester un service en quelques heures. Un pic de fréquentation ? En quelques minutes, l'infrastructure Cloud monte en charge. Et inversement, on déprovisionne tout aussi facilement. Surtout, on ne s'occupe plus du matériel, de l'infrastructure, de sa maintenance, de son déploiement physique ! Les développeurs, les équipes se concentrent sur leur métier, sur leurs compétences.

Le développeur tire naturellement profit du Cloud : nombreux services à disposition, une ouverture toujours plus grande, facilité de déploiement. Les SDK, API, librairies facilitent la vie du développeur que se soit pour des développements from scratch (nouvelles applications) ou des migrations, en 100 % Cloud ou en modèle hybride.

IaaS et « Web Sites » s'utilisent dans une très grande diversité de projets, de développements, d'usages : paiements, gestion et analyse de données, back-end mobile… On peut tout faire, ou presque. Azure fournit des services PaaS qui permettent au développeur de se concentrer sur le code et non sur l'infrastructure.

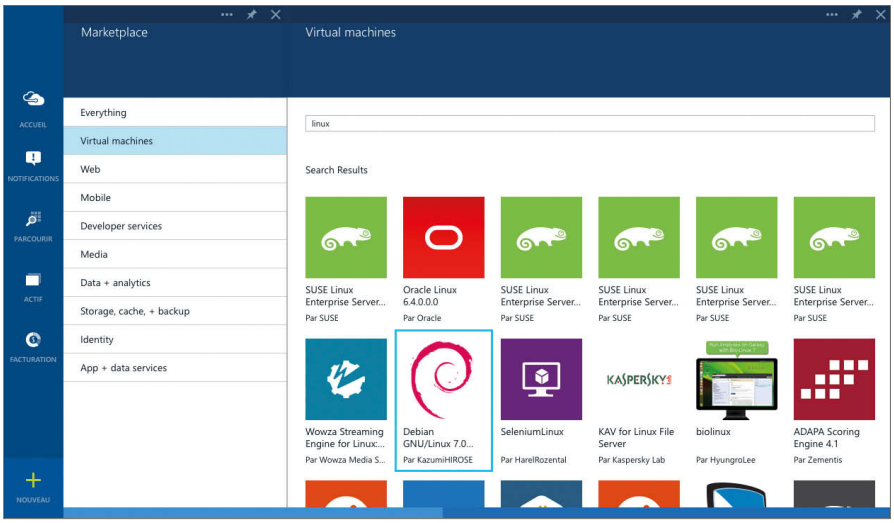
Dans ce cahier spécial Microsoft Azure, le focus a été mis sur des retours d'expérience réels de startups ciblant des services et des marchés très différents. L'un des aspects les plus intéressants est de voir comment les développeurs ont utilisé et intégré des briques Open Source dans leurs architectures IaaS et PaaS.

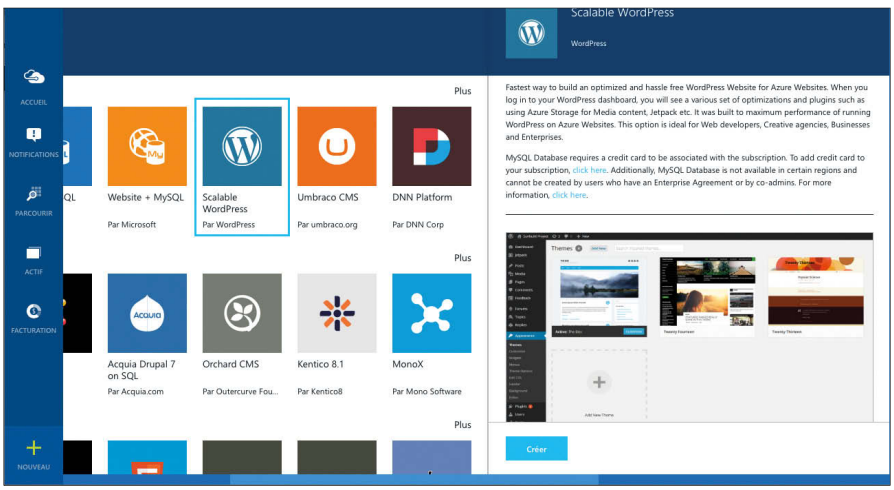
François Tonic
Directeur de la publication
& rédacteur en chef de Programmez !

sommaire

Microsoft Azure : diversité, open source, ouverture

Microsoft Azure héberge et expose aussi bien des technologies et services issus des plateformes Windows et .Net que des environnements non Microsoft. Cette ouverture est primordiale pour pouvoir s'intégrer à un environnement technologique toujours plus hétérogène, mêlant logiciels propriétaires/commerciaux et open source. Il s'agit d'utiliser le meilleur de chaque « monde ». Azure offre aujourd'hui une couverture technique et fonctionnelle très large allant du stockage aux outils de big data, d'analyses à la demande, en passant par les mécanismes hybrides pour connecter les logiciels et données on-premises au monde Cloud.





L'ouverture sur les services PaaS

Azure propose des services PaaS qui permettent au développeur de se concentrer sur le code, sans se préoccuper de l'infrastructure.

Ils aident à créer, à déployer et à exécuter des applications. Pour ce faire, le PaaS utilise des runtimes pour supporter tel ou tel langage et technologie.

Des SDK (kits de développement) permettent aux développeurs de créer des applications Azure et d'intégrer les services Cloud.

Au début, Azure supportait uniquement les langages .Net et Visual Studio.

Mais, aujourd'hui, Azure est très ouvert et supporte de nombreux langages non-Microsoft : Java, PHP, Node.JS, Python, Ruby... Ainsi, si vous avez des applications et projets dans ces langages, vous pourrez les migrer sur Azure et profiter des avantages de la plateforme comme la réplication, la montée en charge automatique, la qualité de service (SLA), etc.

Par exemple, si vous utilisez Java et Eclipse, un SDK Java est disponible ainsi qu'un plug-in pour Eclipse. Azure supporte Tomcat et Jenkins. Ces librairies sont disponibles sur Windows, Linux et OS X. D'autre part, Microsoft et Oracle collaborent étroitement pour faciliter le déploiement et l'usage des serveurs et frameworks Java sur le Cloud : JDK 8, WebLogic.

Même chose pour PHP : le développeur PHP pourra utiliser le SDK dédié et ses outils habituels. Le framework Symfony, un des plus importants frameworks dans l'univers PHP, est disponible sur Azure.

Azure + Linux : le couple parfait

Uniquement PaaS à l'origine, Azure propose une solution offre d'infrastructure (IaaS). Azure Machines Virtuelles permet de créer et d'exécuter des machines virtuelles Windows, Linux et des workloads dédiés (SQL Server, SharePoint, Oracle, Puppet, Chef, Docker, etc.).

Sur la partie Linux, vous pouvez déployer en quelques minutes une distribution Ubuntu Server, SUSE/OpenSUSE, CoreOS, CentOS. Ce sont les workloads proposés par défaut dans le portail Azure. Microsoft travaille beaucoup avec les communautés, et, pour faciliter les publications des machines virtuelles

Open Source pré-packagées, vous disposez de VM Depot.

VM Depot est géré et maintenu par Microsoft Open Technologies. Plusieurs centaines d'images virtuelles sont

Rappel : qu'est-ce qu'Azure ?

Azure est la solution Cloud Computing de Microsoft. Nous y trouvons à la fois des services PaaS (plateforme) et des services IaaS (infrastructure). À cela se rajoutent des fonctions de big data, d'HPC, de cloud hybride, entre autres.
Azure permet de déployer (ou de migrer) des applications, des sites Web, des infrastructures en quelques minutes. Ces services sont disponibles dans le monde entier grâce à 16 datacenters, dont deux en Europe.

Au-delà du PaaS et du IaaS, Azure permet de mettre en place une démarche DevOps grâce à des outils très puissants tels que Release Management, les outils de tests, Application Insight, Visual Studio Online ou encore les fonctions de Build.

accessibles et utilisables sous Azure : Linux, BSD, GitLab, Drupal, Alfresco, Apache, etc. Des images LAMP (Linux, Apache, MySQL, PHP) sont disponibles sur le dépôt.
Vous pouvez rapidement créer votre propre environnement LAMP : http://aka.ms/rf2jiv
Sur VM Depot vous pouvez déposer des images qui peuvent être utilisées depuis le portail Azure.
Si vous ne trouvez pas votre workload, vous pouvez rapidement créer et déployer votre propre environnement, en créer une machine virtuelle Linux, et en y installant toutes les piles techniques nécessaires à vos applications et environnements serveurs. Le IaaS Azure est très souple.
Et vous bénéficiez de tous les mécanismes (activés par défaut ou disponibles en option) de réplications, de load balancing, de montée en charge automatique (auto-scaling).

Web Sites : la diversité du Web à votre portée

Le Web est une des plateformes les plus hétérogènes, avec de nombreuses solutions open source. Tout naturellement, Azure Web Sites supporte cette diversité :
- Node.JS, PHP, Java, Python, HTML 5,
- Git, GitHub pour gérer les sources et les ressources,
- MySQL, MongoDB pour la base de données.

À cela se rajoutent des packages Web prêts à l'emploi : WordPress, Umbraco, Joomla, Drupal… En quelques minutes, vous déployez la solution Web que vous personnalisez ensuite. Idéal pour une startup ou un site Web événementiel à courte durée de vie, mais avec une très forte charge.
Les sites Web hébergés sur Azure bénéficient du load balancing, de la montée en charge automatique (autoscaling), du déploiement continu.
Vous pouvez essayer gratuitement Microsoft Azure. Si vous êtes une startup, vous pouvez bénéficier du programme BizSpark donnant droit, pour une durée de 3 ans, à des ressources Azure, des outils de développement, et aux supports techniques. ⊞

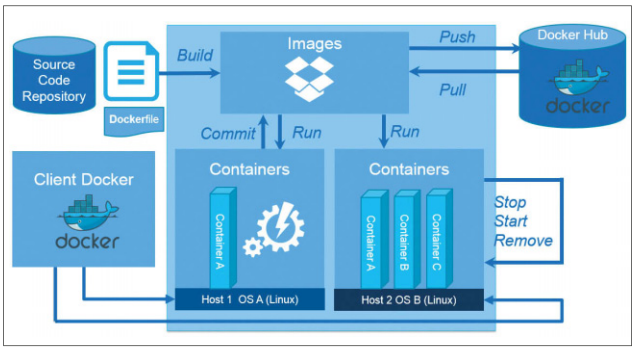
Une seule adresse :
http://azure.microsoft.com/fr-fr/

AVEC DOCKER, LE CLOUD COMPUTING EST-IL SUR LE POINT DE FRANCHIR UN NOUVEAU CAP ?

La virtualisation des machines est le fondement du Cloud Computing tel que nous le connaissons aujourd'hui. Pourtant, il semble que les grands acteurs du Cloud public s'intéressent de plus en plus aux mécanismes de virtualisation à base de conteneurs (ou containers), ainsi qu'aux solutions permettant de les gérer plus efficacement. Parmi celles-ci, le projet Open Source Docker suscite une forte adhésion.
Docker se fonde sur les mécanismes de container LXC, une technologie de virtualisation sous Linux.

Faciliter la portabilité et le déploiement

Si le container d'application offre de multiples analogies avec celui du transfert maritime (empaquetage de tout type d'application et de ses dépendances, exécution sur n'importe quel environnement, isolation…), Docker, lui s'inspire directement de son homologue portuaire en facilitant le chargement et le déchargement de ces fameux containers, sa finalité étant de garantir une cohérence et une simplification des modèles de déploiement. Un container Docker se présente comme un répertoire intégrant les



prérequis liés à l'exécution d'une application. Chaque container est isolé des autres containers qui peuvent s'exécuter simultanément sur le même système. Cette approche offre une meilleure utilisation des ressources et permet d'envisager une plus grande densité d'applications.
Un container est créé à partir d'une image Docker. Docker permet aux développeurs de créer, de personnaliser et de composer des images d'application accessibles en lecture seule et de les télécharger vers des référentiels privés ou publics (ex. : Docker Hub), participant ainsi à la construction de tout un écosystème…
Microsoft propose de déployer des environnements Dockers sur des machines virtuelles Ubuntu ou CoreOS déployées dans Azure et offre un outil Open Source de visualisation des clusters Kubernetes pour gérer des conteneurs Docker à grande échelle.
Côté Windows, en complément du Client Docker, une nouvelle technologie de container sera incluse dans la prochaine version de Windows Server pour laquelle Microsoft proposera également une implémentation Open Source de Docker. Il deviendra alors possible de gérer des applications composées sur la base d'images Linux et Windows depuis le même environnement Docker client. Enfin, les images Docker Hub seront fédérées dans la MarketPlace du nouveau portail Azure. De quoi assurer un futur prometteur à Docker et, qui sait, peut-être transformer le mythe du « Write once, run anywhere » en réalité ? ⊞

Stéphane Goudeau
Cloud Architect
Microsoft France

AGGREGO de Semsoft
Une base de données virtuelle en mode Cloud

Semsoft a été fondé en 2009. Il édite la plate-forme Aggrego qui permet d'exploiter instantanément toutes les données pour des besoins applicatifs ou décisionnels. Aggrego assemble dynamiquement l'information fragmentée dans des silos tant internes qu'externes, sans centralisation des données ni programmation. C'est en fait une base de données virtuelle qui permet d'avoir une vue 360° des utilisateurs, des clients, des produits, des contenus.

Semsoft est un essaimage d'Orange Labs et s'appuie sur 12 ans de R&D développés conjointement avec l'INRIA et le laboratoire de Recherche en Informatique d'Orsay.
L'éditeur permet aux entreprises d'accélérer leur transformation numérique pour innover, mieux opérer et mieux décider, en mobilisant toutes les données disponibles, maintenant. Aggrego réduit par un facteur 10 les coûts et les délais de projets d'intégration traditionnellement mis en œuvre avec des technologies de type ETL, Datawarehouse et MDM.

Concrètement, il s'agit d'une solution SaaS de portail, qui agrège à la demande des données multisources permettant aux utilisateurs fonctionnels de :
- Générer des fichiers de données sur la base de requêtes en langage naturel,
- Enrichir des fichiers ou bases de données à partir de données multisources par simple paramétrage fonctionnel,
- Exposer des données au travers d'APIs configurables, permettant à des applications (portails, etc.) et outils (décisionnels, analytiques, etc.) tiers d'accéder à l'ensemble des données comme si elles étaient présentes au sein d'une même base de données, sans centralisation ni programmation.

L'innovation de la solution réside dans l'automatisation de tous les traitements sur les sources (train de requêtes, jointures, réconciliation…), en se basant sur un paramétrage du sens métier des sources.
Cette innovation apporte les gains suivants :
- Réduction par 10 des temps et délais de projet d'intégration de données par rapport aux solutions traditionnellement utilisées de type ELT/Datawarehouse/MDM,
- Totale agilité pour faire évoluer le périmètre des sources, par simple configuration,
- Autonomie des utilisateurs fonctionnels qui peuvent avec des outils simples et conviviaux « prendre le pouvoir sur les données » cloisonnées en silos pour mieux opérer et décider, sans nécessiter de transformer le SI tout en pouvant exploiter les données externes.

Un large choix technologique

Au départ, Semsoft utilisait une plateforme d'hébergement « classique ». La décision de migrer vers Microsoft Azure s'est faite pour deux raisons : les technologies Microsoft avait un très fort engagement sur le Cloud et la richesse de l'écosystème autour de ces technologies.
Pour Semsoft, ce n'était pas qu'un choix technique à court terme (montée en charge, richesse fonctionnelle, coût, support).

La solution étant vendue à de grandes entreprises soucieuses de la sécurité et de la disponibilité, Azure apporte une crédibilité qu'un autre partenaire n'aurait pas.
Par ailleurs, le choix Microsoft est aussi un choix d'écosystème qui regroupe des services cloud, des partenaires de services, d'autres éditeurs, des solutions de front-end (Office365, PowerBI, Dynamic CRM, etc…).
Le choix s'est donc basé sur une réflexion sur le long terme.

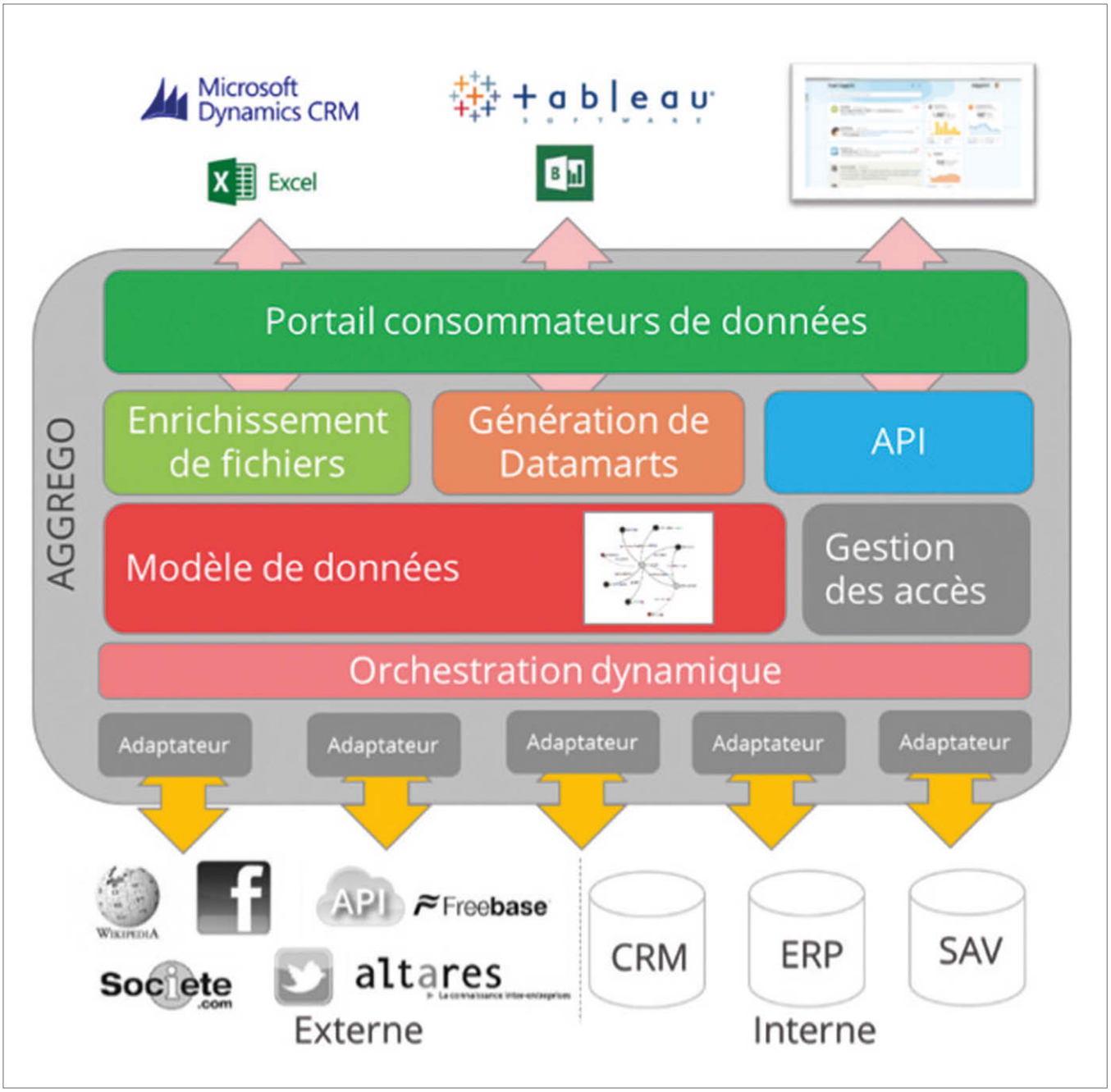
Il était à l'origine contre-nature pour Semsoft ayant une forte culture Java et Open Source, et peu de culture du monde Windows ou .Net.

Comment fonctionne Aggrego ?

Comme dit précédemment, Aggrego est une base de données virtuelle.
Son but est de pouvoir « agréger » dynamiquement des données provenant de différentes sources pour pouvoir les manipuler, les analyser, les intégrer.

AGGREGO fait du mash up dynamique de Web services autour d'un modèle sémantique, métier, configurable.
Il repose sur un hub d'intégration sémantique orchestrant dynamiquement les échanges entre les différentes sources d'information pour en extraire les données utiles et construire, à la demande, des réponses correctes, complètes et homogènes à la vision métier de l'entreprise. Cette vision métier de l'entreprise est matérialisée sous la forme d'un graphe de connaissances (plus précisément une ontologie conceptuelle) explicitant les éléments d'information (c.-à-d. les concepts et les propriétés) exploités au quotidien par les collaborateurs et sur la base desquels sont alignées les différentes sources d'information.
En automatisant, pour chaque demande utilisateur, l'interrogation des sources, ainsi que les processus d'agrégation et de réconciliation des données retournées, AGGREGO se distingue de toute solution existante sur le marché en apportant une agilité sur les données sans équivalent.

L'architecture a été pensée pour accompagner l'évolution des besoins des clients tant sur le périmètre des sources que sur la montée en charge.
Pour permettre cette montée en charge et cette fiabilité, tout en garantissant des niveaux de sécurité importants, Azure a été choisi :
- scalabilité : Azure fournit des modules de gestion de la montée en charge qui ajustent le nombre de serveurs en fonction du trafic et de la charge,
- sécurité : contrôle d'accès du plus simple (HTTPS, IP) au plus spécifique (OAuth)



- fiabilité : supervision et contrôle du SLA au travers d'un portail « user-friendly » nous permettant très facilement d'avoir une vue globale sur l'état et l'utilisation de tous nos services.

Par ailleurs, Azure supporte le système d'exploitation Ubuntu (distribution Linux) qui est l'environnement maîtrisé par les équipes techniques.

Le déploiement de la solution AGGREGO sur Azure a été extrêmement rapide et aucune défaillance n'est survenue.

Semsoft capitalise sur la richesse de l'offre Azure (par exemple : stockage des logs d'AGGREGO, des fichiers de configuration et du cache dans des bases NoSQL telles que MongoDB) pour ajouter des fonctionnalités facilitant la manipulation des données. Enfin, Aggrego se connecte nativement aux protocoles standardisés, comme oData. Les clients peuvent ajouter des sources respectant ce format et c'est encore plus simple lorsqu'elles sont disponibles sur Azure Marketplace !

Le service est facturé à l'usage ou sur abonnement, selon un périmètre de données et d'usage défini. Les abonnements vont de 20 à 100 000 €/an.

Combien ça coûte ?

Le prix dépend globalement du contexte lié au projet client (charge, sécurité, etc.). Cela étant, Semsoft faisant partie du programme BizSpark, la startup bénéficie des prix sans équivalent sur le marché. Dans ce « prix », il faut prendre en compte le support apporté par les équipes de Microsoft. Ce support est commercial, pour potentiellement avoir le support des équipes Microsoft auprès d'un client qui cherche à être rassuré sur la solution provenant d'une start-up.

C'est aussi un support technique, avec l'engagement de la division DX auprès de start-up. Ce support permet de rapidement appréhender les nouveaux services d'Azure et l'ensemble du portefeuille de produits Microsoft. Ce point est particulièrement important pour Semsoft qui n'est pas né avec une culture technique Microsoft.

Salezeo
Réinventer la prospection commerciale

Salezeo a été créée il y a 3 ans. Aujourd'hui, la startup compte une trentaine de personnes, et a levé 1,5 million d'euros auprès du fonds d'investissement Newfund. Dès le départ, la société a misé sur le Cloud Computing et des services SaaS. Le but de Salezeo est simple : proposer une solution de prospection commerciale collaborative innovante en combinant des technologies Open Source et Azure.

L'idée de départ est de briser l'approche traditionnelle de la prospection commerciale et du métier même de commercial et notamment de ne plus être dans des silos étanches, sans échanges.
Salezeo veut améliorer l'efficacité de la prospection commerciale avec un accès plus rapide aux informations, une réduction des saisies de données chronophages et un reporting clair et immédiat, le tout dans une solution mobile et disponible partout.

La donnée au centre de tout

Tout part de la donnée, donc de la base de données. Celle-ci repose sur de multiples sources : réseaux sociaux, communiqués de presse, Wikipédia, Google, géolocalisation, données de l'entreprise, etc.
C'est une base B2B contenant un million et demi de prospects référencés et documentés, 20 000 membres actifs et plus de 50 000 nouveaux contacts chaque mois. Cette base a été construite selon le principe du crowdsourcing.

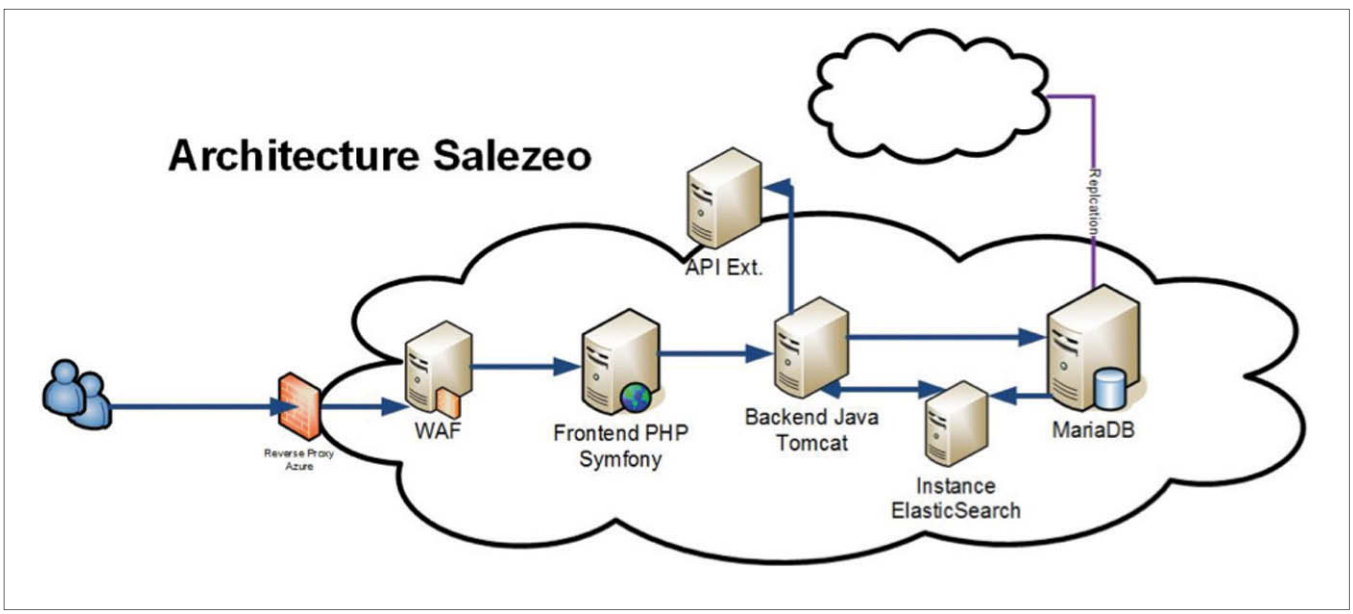
Le crowdsourcing est une approche collaborative pour créer du contenu et collecter des informations. On mutualise ainsi les données provenant des membres pour alimenter la base de données et compléter les informations existantes. Chaque membre échange en ligne, partage avec la communauté, un peu comme un wiki. Salezeo est une plate-forme d'échanges avec des mécanismes de Big Data comme la collecte permanente de contenus et de données provenant du Web. Il s'agit de capter l'empreinte numérique des prospects, des entreprises.
L'idée est de proposer aux commerciaux des informations plus complètes et actualisées afin de mieux cibler et de mieux comprendre les besoins du client et être ainsi plus efficace, plus pertinent dans la démarche commerciale et le développement de nouvelles activités.

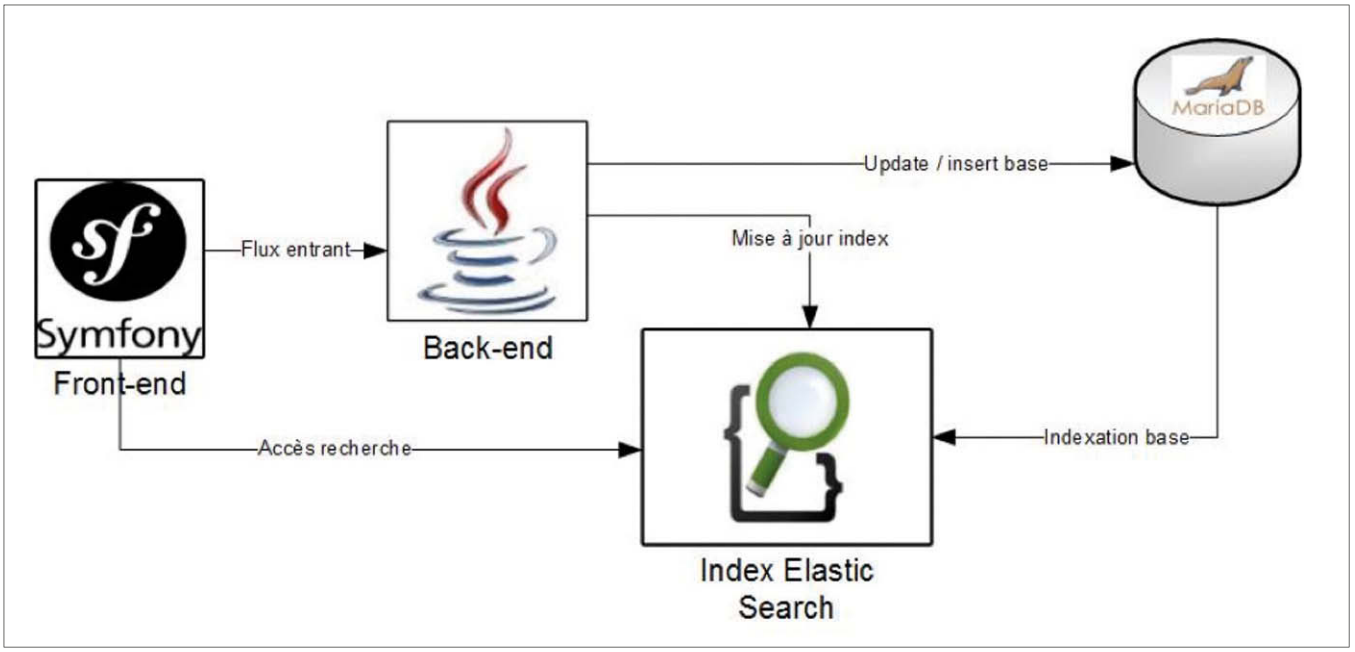
Les principaux enjeux techniques du service concernent la collecte, le traitement et l'accès à de grands volumes d'informations multi-sources.

Le Cloud et l'open source : une flexibilité nécessaire

Dès son lancement, Salezeo avait l'ambition de construire une architecture logicielle et système suffisamment flexible, notamment pour supporter un déploiement à l'international et une forte croissance des volumes de données à traiter, tout en conservant une capacité d'évolution des technologies utilisées et des coûts adaptés.

La solution repose sur des briques open source pour construire des architectures SaaS. La partie front-end s'appuie sur le Framework PHP, Symfony, un choix dicté principalement par les compétences disponibles et la richesse de la communauté en France. Le back-end qui est au cœur du traitement et de la valorisation des données, utilise Java (JEE 1.7). La couche serveur Java est de l'Apache sur lesquelles on y trouve des API internes et externes. L'information collectée alimente ce moteur interne basé sur des algorithmes dédiés et des moteurs





d'apprentissage (Machine Learning). La base dont le volume croît rapidement possède plusieurs dizaines de millions de lignes et est beaucoup sollicitée pour les analyses et les traitements des données.

Le choix de la base de données s'est porté rapidement sur MariaDB, un fork de MySQL, notamment pour ses performances en termes de réplication de base et la possibilité de mettre en œuvre des clusters de base de type Galera pour répondre aux problématiques de montée en charge.

L'un des points majeurs pour ce service est l'accès à la donnée, que ce soit pour le moteur de recherche mis à disposition du front-end ou pour le traitement de la donnée. Pour ce faire, Elastic Search (basé sur Lucene — fondation Apache), associé à Kibana pour visualiser les informations, a été mis en place pour un accès rapide à la donnée, sa flexibilité face à une montée en charge et pour libérer la base de données des accès en lecture. La particularité d'Elastic Search est d'utiliser une architecture distribuée et de s'interfacer avec des bases SQL ou NoSQL. Il supporte REST et JSON. Les performances de l'indexation sont une des problématiques des développeurs.

Les instances utilisées dans cette architecture sont des Machines Virtuelles OS Linux (Debian) dont les performances doivent pouvoir évoluer continuellement pour répondre à la montée en charge de la plateforme. Le tout fonctionne sur les services IaaS d'Azure depuis presque 3 ans.

Ce choix a été dicté par la robustesse de l'offre et le sérieux de la plateforme. Il fallait aussi un Cloud mondial capable d'assurer la disponibilité et les performances partout dans le monde ; en Europe dans un premier temps et rapidement sur d'autres continents. Salezeo est entré dans le programme Bizspark+ de Microsoft. Ce programme accompagne les sociétés à fort potentiel, identifiées comme telles par Microsoft, en proposant support technique et ressources pour accélérer leur croissance.

La performance au cœur des préoccupations

Comme souvent, Salezeo a mis en place des instances de production et de préproduction sur Azure.
La préproduction sert à valider les modifications de codes et l'ajout de nouvelles API des partenaires externes.
Ce n'est qu'après la phase de validation que le déploiement des évolutions se fait en production.

Pour les équipes techniques, les enjeux de performances sont importants, particulièrement en ce qui concerne la base de données et les performances d'écriture. La solution de stockage temporaire basée sur des disques SSD a permis de répondre à cette exigence.

L'infrastructure doit être très flexible sur le front-end et sur le back-end. L'autoscaling est une des fonctions utilisées pour assurer la montée en charge et l'automatisation.

Le loadbalancing est tout aussi crucial. C'est pour cela que Salezeo utilise 2 Clouds chez 2 fournisseurs différents avec des fonctions de backups et de réplication de données. L'architecture Open Source et la non-adhérence technologique sont les deux éléments clés pour pouvoir basculer d'un fournisseur à un autre immédiatement et sur la même architecture. En cas de panne, Salezeo bascule sur le fournisseur de secours.

SLA

La double architecture mise en œuvre par Salezeo assure un niveau de service de haute disponibilité et permet le déploiement d'instances pour chaque brique Open Source déployée (base, indexation, back-end...)
La supervision de la plate-forme est réalisée avec un outil Open Source, Icinga, et le support de la partie IaaS est assurée par Microsoft, avec un support on-line réactif.

Combien ça coûte ?

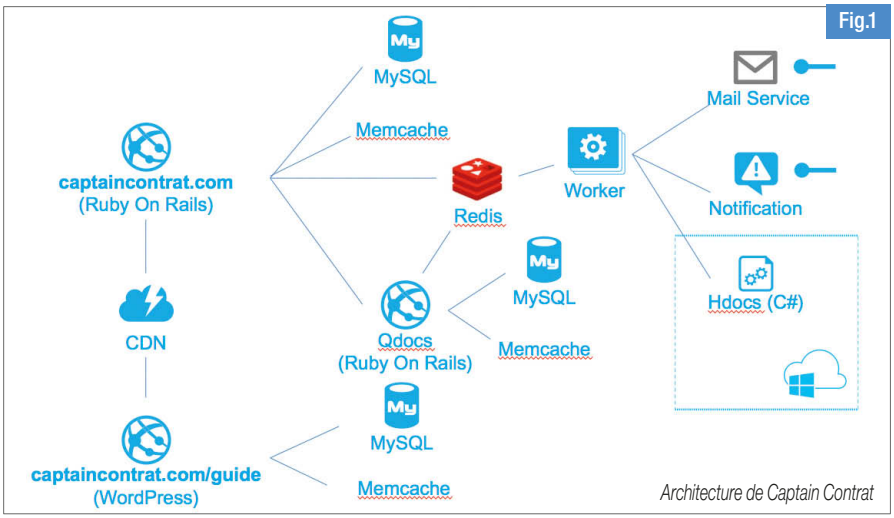
Chaque plate-forme (production et préproduction) représente un coût d'environ 1 100 €/mois.

Ressources

Pour suivre Salezeo :
http://entreprise.salezeo.com
https://twitter.com/Salezeo
https://www.linkedin.com/company/salezeo
https://www.facebook.com/Salezeo

Captain Contrat
Le numérique au service du juridique

Captain Contrat est un site Internet sur lequel des entrepreneurs/sociétés peuvent acheter des documents juridiques à travers une mise en relation avec un avocat à un prix abordable.



La startup a développé un algorithme à destination des avocats qui permet d'automatiser une partie de la production de documents juridiques et ainsi de baisser les prix. Le logiciel comprend les réponses de l'entrepreneur à différentes questions et est capable de modéliser de 80 % à 100 % du document. Ainsi l'avocat intervient tout de suite après que le logiciel ait généré une première version du document et ne se concentre que sur la partie où sa valeur ajoutée est la plus forte (le conseil juridique et la personnalisation avancée du document). Ainsi, les honoraires proposés pour la rédaction des documents juridiques par des avocats sont également plus faibles.

Quand le numérique aide l'avocat

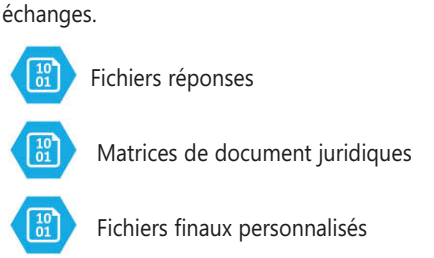
En travaillant avec des cabinets d'avocats sur la façon dont ils rédigent des documents juridiques pour leurs clients, Captain Contrat a constaté des structures communes dans la façon de collecter les informations (série de questions posées au client) et de mobiliser ces informations pour rédiger le document final. De nombreux éléments relèvent du secrétariat et il est possible de l'automatiser. L'innovation réside dans l'algorithme pour chaque document. Ainsi, l'avocat ne fait plus de secrétariat juridique et peut se concentrer sur le conseil juridique.
Captain Contrat a participé à l'accélérateur Microsoft Ventures ce qui a permis un suivi des développements du service SaaS.
L'objectif de ce logiciel SaaS est de permettre à tout profil, y compris et en particulier des personnes n'ayant pas de compétences en développement informatique, de créer leurs propres documents automatisés et de définir des arbres de décision pour la génération automatique de ces documents.

L'architecture

Pour concevoir la plateforme, Captain Contrat a mis en place une architecture sous forme de services. Deux applications Ruby On Rails hébergées dans des machines virtuelles Linux assurent la partie Web pour le site et le questionnaire dynamique. Les bases de données SQL ont été externalisées dans un service dédié, qui assure la réplication et la redondance. Un CDN pour les assets et des services de cache ont été mis en place pour la rapidité et les performances. Un Worker Azure, déployé en quelques clics(1) via le SDK Azure pour Visual Studio, et quelques services de stockages et échanges de messages assurent la communication entre les applications Ruby et .NET Fig.1.

La personnalisation des documents

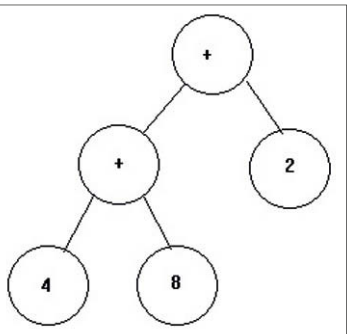
L'utilisateur accède au service par le site principal https://captaincontrat.com/, choisit un document et fournit les détails sur son entreprise.
L'outil de collecte d'informations est une application Web Ruby On Rails qui à la fin du questionnaire exporte un fichier XML contenant l'ensemble des réponses vers un des 3 Blob Storage Azure utilisés pour les échanges.



3 Blob Storage Azure pour les échanges de fichiers

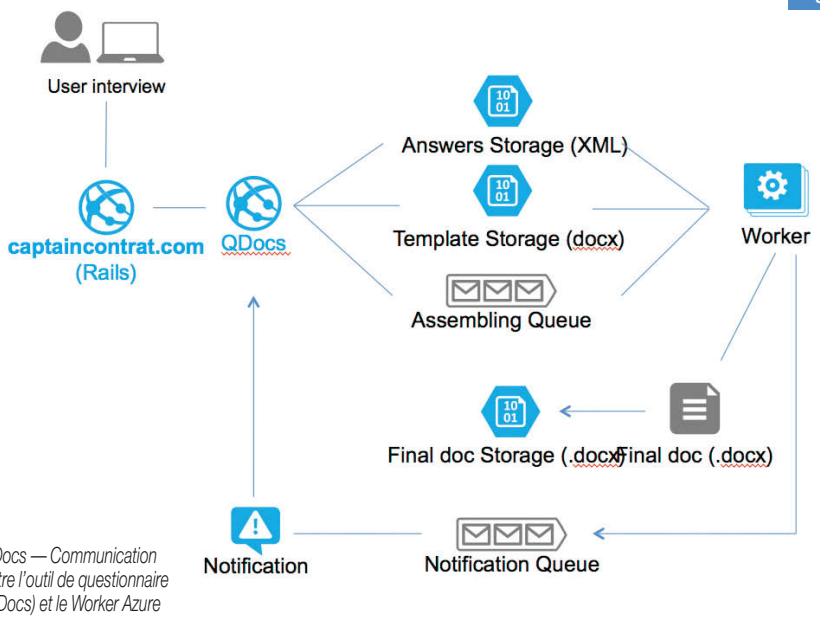
Un message contenant la tâche à exécuter est envoyé dans une Queue de type Service Bus et le fichier matrice est stocké dans un

PRÉCISIONS SUR L'AST



AST (abstract syntax tree), est une structure de données qui va contenir une expression issue d'un langage de programmation en la décomposant en tokens et qui va respecter la syntaxe de ce langage. Cet outil est principalement utilisé par les compilateurs. Exemple : dans beaucoup de langages, nous pouvons réaliser une addition, celle-ci possède la forme 2 + 2 (nombre + nombre). Si nous avons 2 + (4 + 8), nous avons une priorité d'opération (d'abords le 4 + 8, puis le 2 + (12)), il faut la représenter sous une forme qui permet au compilateur de faire son travail.
Cette représentation va permettre au compilateur de comprendre qu'il doit d'abord traiter 4 + 8, et d'utiliser ce résultat pour faire l'opération suivante (result + 2).

Fig.2



HDocs — Communication entre l'outil de questionnaire (QDocs) et le Worker Azure

second Blob Storage. La communication entre Ruby On Rails et les services Azure se fait avec le SDK pour Ruby disponible sur Github(2) qui permet l'upload/récupération de fichier et l'envoi de message en assurant la communication avec les APIs Azure pour les services Blobs/Queues Storage et Service Bus Queues Fig.2.

Un Worker Azure requête la queue régulièrement dans l'attente d'une tâche à exécuter. Quand une tâche est disponible, il récupère les informations des documents à assembler, fournis dans le message.

La sauce secrète de Captain Contrat se trouve dans ce service qui à partir d'une matrice au format Docx et des réponses obtenues, personnalise un document juridique aux besoins du client. Le générateur de documents est développé en interne en C#.

Lorsqu'un nouveau questionnaire est créé dans l'interface Web, la matrice du document est mise à disposition de l'assembleur dans le stockage dédié aux matrices. Le générateur peut ensuite aller récupérer ces matrices ainsi que les fichiers XML de réponses dans les stockages pour commencer l'assemblage et appliquer les algorithmes.

Une bibliothèque de classes C# a été développée, qui donne un fichier .dll une fois compilé, et fait appel à plusieurs autres librairies disponibles en Open source et fournies par Microsoft, principalement :

- Open XML SDK 2.5(3)(4) permettant la manipulation de document office. Rendu Open Source en juin 2014(5).
- .NET Compiler Platform(6) (« Roslyn ») permettant des traitements complexes (Roslyn permet entre autres de pouvoir générer des AST à partir d'un code C#, d'interpréter du code C# stocké sous forme de string au runtime...).

Ces librairies sont installées via le système Nuget package(7), disponible dans Visual Studio, qui permet le téléchargement de librairies externes(8) et la gestion des références au sein du projet.

Dans le fonctionnement de l'assembleur, Open XML SDK est indispensable. Cette librairie permet la manipulation des nodes XML et attributs au sein de document au format Office Open XML(9) (docx, xlsx, pptx...) et assure la validation du schéma XML(10) en sortie. Elle est difficilement remplaçable au vu de sa compatibilité avec le format de fichier et sa puissance/simplicité d'utilisation.

Roslyn fournit un compilateur de codes sources C# et des API d'analyse de codes. Il est utilisé pour évaluer les expressions et la logique stockée dans les matrices. Tous les traitements associés sont développés sous forme de module remplaçable pour pouvoir ajouter ou supprimer des fonctionnalités ou faire évoluer la librairie si nécessaire.

Les matrices de document au format Office 2013 contiennent des marqueurs de type Content Controls (défini par la norme ECMA 376(11), qui précise les conventions de format de fichier Open Office XML(12). Ces marqueurs indiquent les blocs où des traitements vont intervenir et la manipulation a appliqué (Ex : l'insertion d'une valeur donnée par le client, Calcul de l'âge, Conversion de nombres en lettres...). Les algorithmes d'assemblage sont appliqués sur la matrice à partir des informations fournies. Le fichier assemblé (au format .docx) est uploadé sur le stockage final et une notification est envoyée à l'application principale pour qu'elle le récupère et le communique à l'avocat qui va relire et ajouter sa touche finale.

Le développement

Des heures de lecture de la documentation MSDN, des normes et des ressources du site OpenXmlDeveloper(13) ont été nécessaires pour comprendre le format Office XML, le manipuler et développer un plug-in Office. Des développements sont encore en cours avec le .NET Compiler Platform (« Roslyn »), pour une partie du traitement logique des documents.

Pour le développement, Visual Studio 2013, Office 2013 et Azure SDK pour Visual studio sont utilisés. Ces outils ont été obtenus dans le cadre du programme Bizspark Plus(14). Le dernier permettant la simulation en local de l'environnement Azure et le déploiement de l'ensemble dans le Cloud.

Combien ça coûte ?

Aujourd'hui la plateforme consomme environ 200 € du crédit par mois pour quelques services : machine virtuelle, stockage, service Bus, CDN. Captain Contrat a été finaliste du concours Frenchweb /Azure 2014 ce qui donne accès à des ressources pendant 24 mois.

(1) http://msdn.microsoft.com/en-us/library/azure/jj149831.aspx
(2) https://github.com/Azure/azure-sdk-for-ruby
(3) https://github.com/OfficeDev/Open-XML-SDK
(4) http://msdn.microsoft.com/fr-fr/library/office/cc471858%28v=office.15%29.aspx#BKMK_Introduction
(5) http://blogs.office.com/2014/06/25/open-xml-sdk-goes-open-source/
(6) http://msdn.microsoft.com/fr-fr/vstudio/roslyn.aspx
(7) https://www.nuget.org
(8) http://msdn.microsoft.com/fr-fr/magazine/hh547106.aspx
(9) http://fr.wikipedia.org/wiki/Office_Open_XML
(10) http://fr.wikipedia.org/wiki/XML_Schema
(11) http://www.ecma-international.org/publications/standards/Ecma-376.htm
(12) http://msdn.microsoft.com/fr-fr/library/office/gg607163%28v=office.14%29.aspx
(13) http://openxmldeveloper.org
(14) http://www.microsoft.com/BIZSPARK/plus/default.aspx

Alkemics
entre algorithme et machine learning

La startup Alkemics propose une plateforme de normalisation et d'enrichissement des données produits aux acteurs de la grande distribution. Grâce à cette solution, l'éditeur propose des services de moteur de recherche, substitution et recommandation basés sur la forte compréhension des données produits et des interactions des utilisateurs avec ces produits.

L'équipe technique regroupe des profils complémentaires :
- des profils orientés mathématiques appliquées et statistiques, pour le datamining et le machine learning,
- des profils très techniques sur les problématiques big data, notamment sur la gestion des infrastructures, bases de données et outils.

Algorithme et machine learning

Au cœur du fonctionnement de la plateforme Alkemics, nous trouvons des algorithmes de Traitement du langage (NLP) et du Machine Learning (ML) pour traiter les importants volumes de données produits et utilisateurs.

L'approche est basée sur l'utilisation d'un graphe sémantique pour représenter à la fois les produits, les utilisateurs et surtout les relations qui les lient.

L'utilisation d'un graphe, à la fois pour le modèle de données et son implémentation permet d'avoir une vision très limpide des relations. Le marketing est avant tout une histoire de relations, cette approche est donc particulièrement pertinente pour les clients.

Ce graphe est mis à disposition des marques en tant que plateforme SaaS, les besoins en capacité de traitement des requêtes évoluent donc au gré des campagnes marketing de ces utilisateurs, d'où la nécessité d'une infrastructure Cloud.

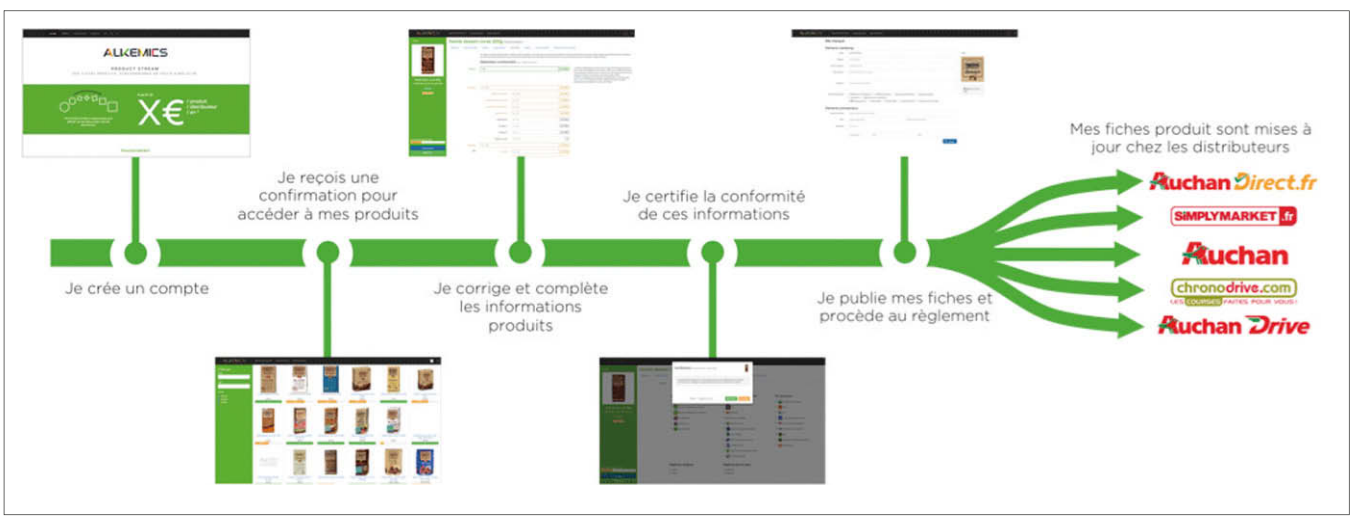
Comment ça marche ?

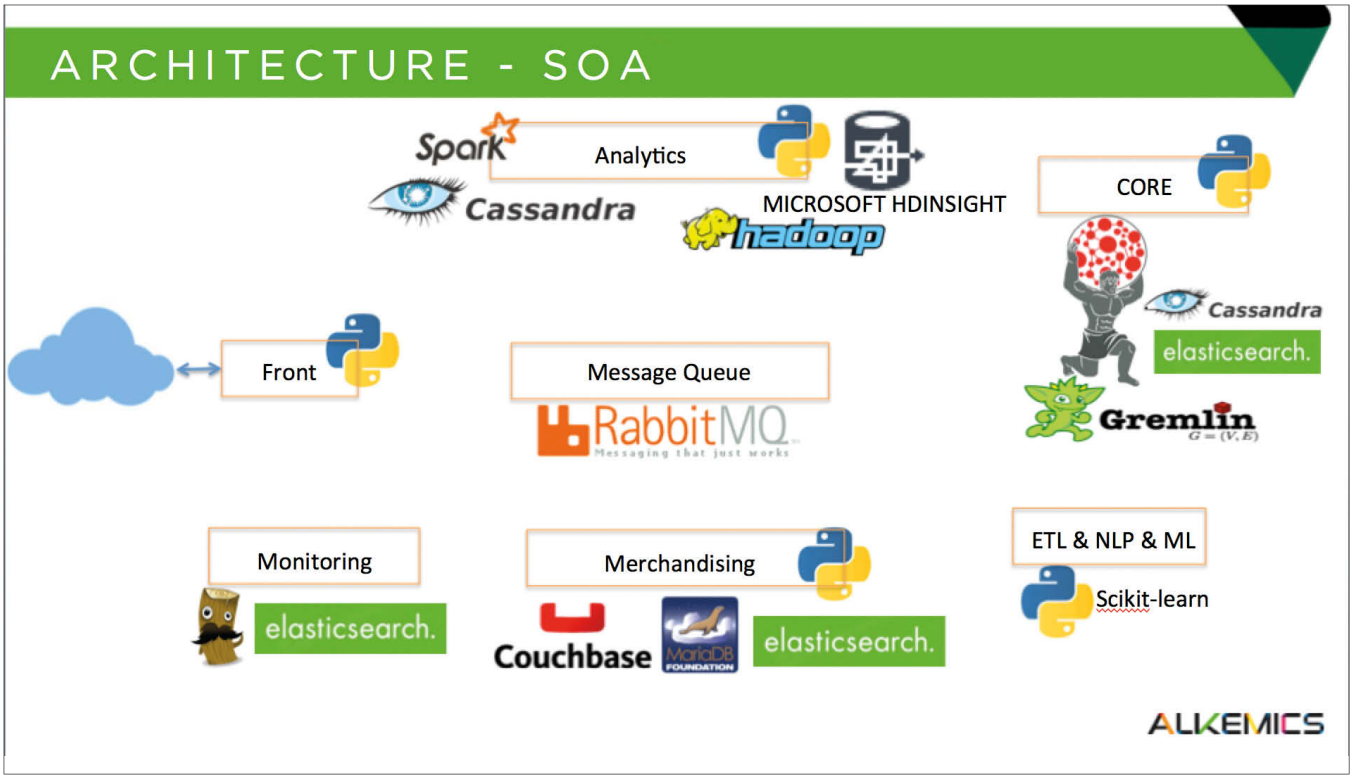
L'infrastructure de production est une architecture orientée services, constituée d'une cinquantaine de composants, communiquant grâce à un broker RabbitMQ et des requêtes HTTP. Cette architecture permet beaucoup de flexibilité en termes de choix technologiques pour l'implémentation du back-end.

Alkemics a pu ainsi choisir et faire évoluer chacun des services indépendamment les uns des autres pour répondre au mieux aux besoins de développement. Cette architecture a l'avantage de permettre d'adapter dynamiquement les ressources pour chacun des services sollicités, pour mieux répondre aux besoins réels de traitement. C'est la flexibilité du Cloud.

La plupart des services sont historiquement écrits en Python. En effet, l'utilisation de Python donne accès à de très nombreuses librairies open source (ntlk, scikit-learn, scipy, numpy) pour le traitement du langage naturel et le Machine Learning. À mesure que la plateforme évolue, les équipes techniques intègrent de plus en plus de composants Go, pour les microservices et en Scala pour interagir avec Spark.

Les APIs sont exposées en HTTP via des serveurs Web nginx ou en AMQP via RabbitMQ. Ces services communiquent avec des backends variés en fonction des besoins :
- Couchbase pour les vues matérialisées et le cache distribué, primordial pour conserver les performances raisonnables alors que la volumétrie de données augmente,
- ElasticSearch pour les moteurs de recherche,
- Cassandra pour le stockage de timeseries d'analytics, les vues matérialisées de matrices pour la recommandation, un





event store pour les événements générés sur notre plateforme et le backend de stockage pour l'ensemble des informations normalisées d'interactions des utilisateurs.

En particulier, la brique Core est constituée essentiellement de Titan, une base Graphe distribuée construite sur un backend de stockage Cassandra et un backend d'indexation ElasticSearch.
L'utilisation du graphe donne une flexibilité extrême en termes de modèle de données tout en permettant des requêtes normalisées. Pour des raisons de performance, la R&D étudie depuis peu la possibilité d'utiliser GraphX sur un cluster Spark+Cassandra.

Enfin, le pipeline d'analytics est constitué de jobs PIG qui sont exécutés sur un cluster HDinsight (solution Hadoop-as-a-service disponible sur Azure). Les données préprocessées sont alors ingérées dans Cassandra pour le reporting, les analytics et le datamining interactif.
Cette utilisation d'un cluster hadoop-as-a-service permet de prétraiter et filtrer les forts volumes de données reçus de manière complètement scalable et en temps constant, sans nécessiter la maintenance d'un cluster en permanence.
Quotidiennement, le cluster HDInsight est créé et détruit. Il est utilisé environ 3h pour effectuer les traitements batch sur les données.

La plateforme permet aux marques de contribuer à l'enrichissement des fiches de leurs produits sur les sites des distributeurs. Cet enrichissement servira alors à améliorer l'expérience utilisateur sur le site d'ecommerce : fiches produits détaillées (images, labels de qualité, valeurs nutritionnelles, etc...), plus grande pertinence des résultats des moteurs de recherche et des recommandations.

Le processus d'échange peut être automatique ou manuel selon la maturité des systèmes d'information des marques.
Par exemple, nous mettons à disposition un portail web sur lequel les marques peuvent se connecter pour procéder aux échanges.
Nous leur proposons alors d'importer les données déjà présentes dans leurs bases de données dans les formats standards GS1 d'échange de données produit.

Une fois un premier import effectué, nous permettons aux marques d'enrichir la donnée initiale avec les éléments obligatoires pour les distributeurs ainsi qu'avec des informations marketing afin de mieux mettre en valeur leurs produits : visuels, labels et certifications, histoire du produit, etc...
La marque peut alors déclencher le transfert de ses fiches au distributeur.

Une fois cette étape effectuée,
les données brutes sont normalisées par nos algorithmes de façon à les rendre exploitables par nos APIs.
Nous transmettons alors les données brutes aux distributeurs pour alimenter leur base produit et utilisons les attributs normalisés pour faire fonctionner nos webservices (moteur de recherche, substitution, etc...).

Ressources

- Partitioning sur graphe : http://www.cs.utexas.edu/users/inderjit/public_papers/kdd_bipartite.pdf
- Partitioning (papier payant) : http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.8969
- Mapreduce sur titan : http://s3.thinkaurelius.com/docs/titan/0.5.2/hadoop-performance-tuning.html#d0e14203
- Graphx (papier) : https://amplab.cs.berkeley.edu/wp-content/uploads/2013/05/grades-graphx_with_fonts.pdf
- Faire de la théorie des graphes de manière scalable sur un property graph : https://www.kent.edu/sites/default/files/TR-KSU-CS-2010-01.pdf
- Recommandation de tags par graph ranking : http://www.researchgate.net/publication/221300417_Personalized_tag_recommendation_using_graph-based_ranking_on_multi-type_interrelated_objects/file/60b7d521eb53e9d537.pdf

Movidone
Du mobile, du IaaS, du code natif

Jeune startup Strasbourgeoise, les équipes de Movidone ont eu une idée géniale : transformer en terminal de paiement son smartphone. Comment ? Avec une simple app : EasyTransac Pro.

Avec cette app mobile, Movidone mise sur le marché professionnel, demandeur de ce genre de solution pour minimiser les investissements matériels et pour éviter de multiplier les terminaux et les fournisseurs.

Le principe d'EasyTransac Pro est simple : l'application offre la possibilité aux commerçants, professions libérales ; autoentrepreneurs ou associations de recevoir instantanément des paiements via un terminal mobile comme en point de vente, de manière simple, sécurisée et efficace.

La mobilité permet l'innovation

Le « tout connecté » est une réalité pour les utilisateurs et les entreprises. Et le paiement mobile se développe, pas seulement avec les technologies de type NFC. Pour Movidone, le principe de l'app est simple à comprendre : EasyTransac est au paiement ce que le smartphone est à la cabine téléphonique.

EasyTransac PRO permet aux entreprises de profiter des avantages d'un TPE(1), sans les contraintes que cela implique. Combiné avec les atouts du Cloud d'Azure (PCIDSS complient et un SLA proche de 100 %),

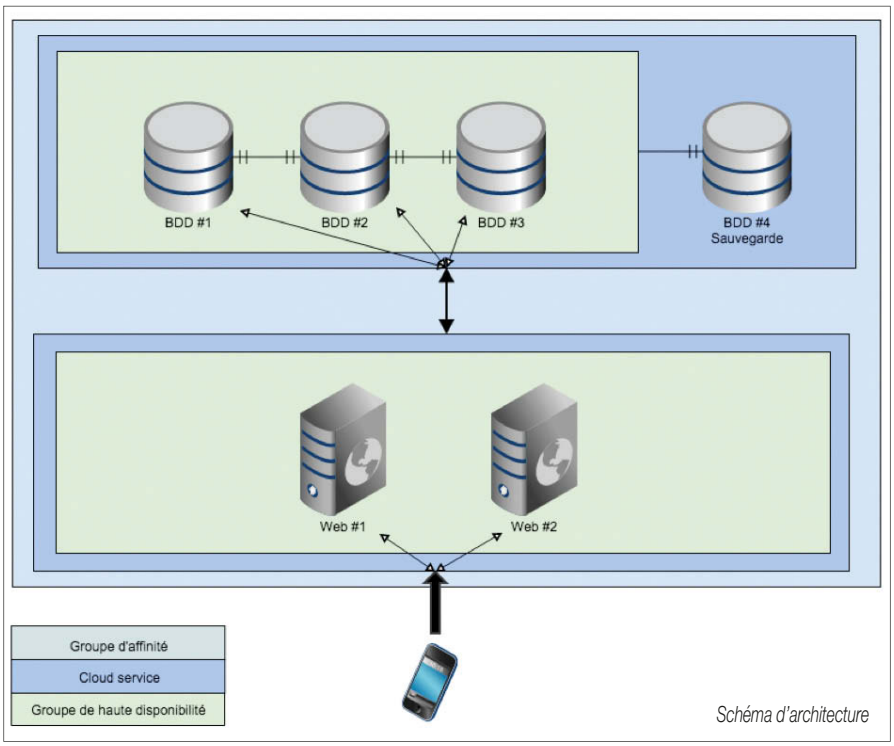
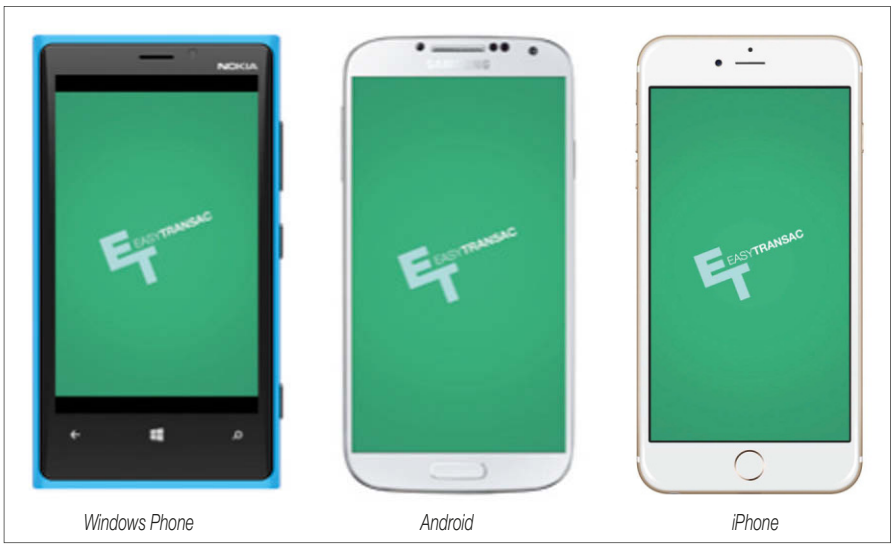


Schéma d'architecture

EasyTransac PRO offre à ses clients un moyen de paiement sans fil extrêmement fiable et sécurisé.

Architecture et fonctionnement

Dans son fonctionnement, EasyTransac PRO repose sur trois éléments : l'hébergement, le Web service et les applications.



Windows Phone *Android* *iPhone*

L'hébergement

L'hébergement choisi pour le projet EasyTransac est un hébergement de type Cloud de bas niveau selon le modèle de service IaaS(2.) En effet, il y avait des prérequis importants en termes de sécurité et de haute disponibilité, car nous traitons des transactions bancaires et il était important de maîtriser l'ensemble des applications fonctionnant sur la machine et d'avoir une maîtrise totale du middleware (côté applicatif). Autre nécessité : que l'hébergeur soit certifié PCI DSS de niveau 1 afin d'assurer un maximum de sécurité sur l'ensemble de la chaîne.

Le choix s'est porté vers la plateforme de Microsoft Azure. Movidone bénéficie depuis plus d'un an, du programme BizSpark Plus. La plateforme répond aux exigences définies : SLA(3) de 99,9 %, des datacenters répartis partout dans le monde, du stockage géo-redondé et est certifiée PCI DSS de niveau 1.

Pour commencer, les équipes ont créé un groupe d'affinité afin d'optimiser les

performances de l'ensemble de l'infrastructure. Cela a pour conséquence de « rapprocher » physiquement les services associés à ce groupe. Puis de déployer l'ensemble des ressources associées à ce projet dans ce groupe d'affinité.

Le projet est composé de deux « Cloud Services » distincts :
- un « Cloud Service » Web.
- un « Cloud Service » base de données.

Ces deux services ont été construits en respectant les « bonnes pratiques » d'Azure en matière de SLA.
C'est-à-dire qu'ils sont chacun composés d'au moins deux machines identiques qui se trouvent dans un « groupe de haute disponibilité ».
Cela assure qu'en cas de maintenance de la plateforme, les services associés continuent de fonctionner parfaitement sans interruption de services.

Afin d'assurer l'équilibrage de charge, Movidone a créé pour chaque service critique (par exemple Apache) un point de terminaison directement connecté au « Cloud Service » de la machine. Ainsi, la charge est automatiquement partagée entre les serveurs.

Les développeurs profitent également du système de « sondes » d'Azure, qui permettent de vérifier la disponibilité d'une machine au sein du « groupe de haute disponibilité(4) » et d'agir en conséquence. L'ensemble des informations nécessaires pour la bonne compréhension de ce point est disponible sur :
http://azure.microsoft.com/fr-fr/documentation/articles/virtual-machines-load-balance/

Chaque machine est également équipée d'outils de monitoring (monit, Zabbix) ainsi que d'outils de sécurité.
Voyons à présent de quoi est composé en détail chaque « Cloud Service » :

Le « Cloud Service » Web
Le « Cloud Service Web » est composé de deux serveurs frontaux identiques, composés de 2 cœurs et de 3,5 Go de RAM. Les ports 80 et 443 d'Apache sont loadbalancés et équipés d'une sonde vérifiant la disponibilité des services.

Le « Cloud Service » de base de données
Le « Cloud Service de base de données » est plus complexe : il s'agit d'un cluster de base de données MariaDB (fork de MySQL). Il est composé de 4 machines identiques, équipées de 4 cœurs et de 7 Go de RAM chacune. Trois de ces machines sont loadbalancées sur le port défini par le service de base de données de la même manière que le « Cloud Service web », c'est à dire à l'aide de sondes. Ces trois machines sont en réplication « Master/Master ». La dernière machine est utilisée en dehors du loadbalancing afin de ne l'utiliser qu'en réplication « Slave ». Cette machine est dédiée à des tâches précises telles que des sauvegardes de la base de données, etc.

Le Web service
Le cœur de la solution EasyTransac repose sur un Web service REST qui permet à une application tierce d'envoyer et de recevoir des informations. La conception logique de cette application repose sur un « Framework MVC » codé en PHP.

Les applications
Les applications mobiles sont écrites en code natif pour un souci de performance et d'optimisation (Objective-C pour iOS, Java pour Android et C# pour Windows Phone).

Ces applications utilisent et communiquent avec le Web service REST qui se trouve dans « Cloud Service » Web.

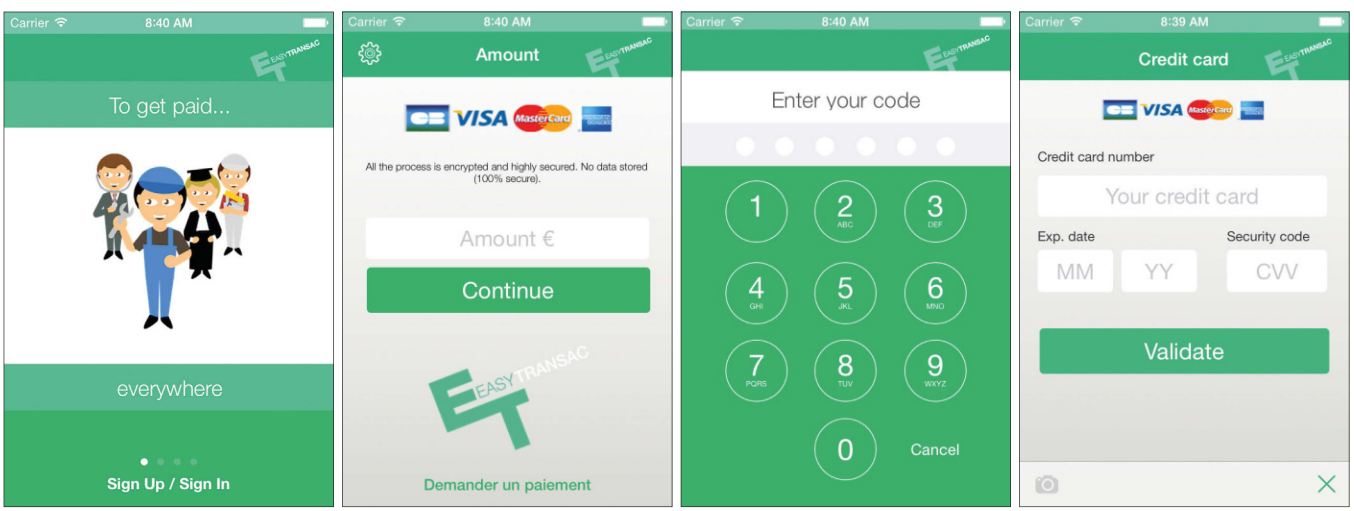
Un coût à la demande
Avec le Cloud Computing, les coûts se modulent selon les besoins et la consommation effective. La facturation dépendra donc des services utilisés et les ressources (ressources consommées, services provisionnés). L'infrastructure pour EasyTransac Pro revient à environ 979 €/mois, pour 6 instances.

Dans le détail :

Machine	Nombre	Prix
Standard Linux A3	4	531.90 €
Standard Linux A2	2	265.96 €
TOTAL	**6**	**979.86 €/mois**

Pour aller plus loin
Calculateur de prix Azure :
http://azure.microsoft.com/fr-fr/pricing/calculator/?scenario=virtual-machines
Haute disponibilité Azure :
http://azure.microsoft.com/fr-fr/documentation/articles/virtual-machines-manage-availability
MariaDB en détail :
https://mariadb.com/
EasyTransac :
http://www.easytransac.com/

(1) TPE : Terminal de paiement électronique
(2) IaaS : Infrastructure as a service
(3) SLA : Service Level Agreement
(4) Groupe de haute disponibilité : couramment appelé availability set sur la plateforme Azure

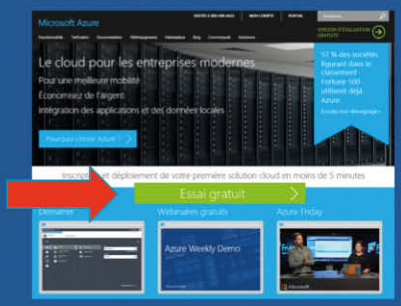


Ressources

Un mois d'essai offert

http://azure.com

150 € de ressources offertes
Sans engagement



Microsoft BizSpark

L'offre destinée aux startups

http://www.microsoft.com/bizspark/

BizSpark

115€ /mois
x5 membres
x3 ans
= 4175€ d'équivalent cloud

BizSpark Plus

3 723,50 € /mois
x1 an
= 44 682 € d'équivalent cloud

Abonnés MSDN

http://aka.ms/azurepourmsdn

Activez vos bénéfices Azure jusqu'à 115 € de ressources mensuelles offertes

Formation et Accompagnement

Microsoft Virtual Academy

http://www.microsoftvirtualacademy.com/

La plateforme de formation technique…100% gratuite

Pépinière Microsoft Azure

http://aka.ms/pepiniereazure

Un accompagnement technique et business gratuit et illimité pour tous les projets cloud

Microsoft Ventures

http://aka.ms/VenturesParis

3 mois de coaching business et technique pour faire passer votre startup de l'idée à l'exécution

Suite page de la page 31

3 - On met le focus sur le process de rang 0 (car c'est lui qui dispose de `full_domain`),
4 - On double-clique sur `full_domain` que l'on va caster en `double[1000]`,
5 - Tools -> Visualize Fig.1.

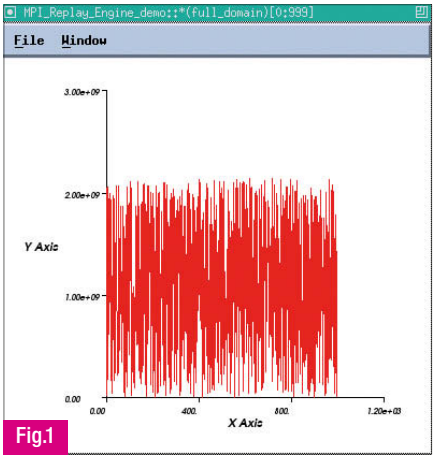


Fig.1

Les valeurs sont plus ou moins aléatoirement réparties entre ~0 et ~2000000. Enlevons le breakpoint que nous avons précédemment établi. Maintenant, nous allons nous intéresser à la question suivante : les tris partiels ont-ils été correctement effectués par chacun des process ? Il s'agit donc de contrôler visuellement sur le rang 0 la variable `full_domain`, juste après que celle-ci ait reçu chacun des tableaux partiels triés :

1 - Positionnez un breakpoint après l'exécution de `MPI_Gather(...)` (L89 par exemple),
2 - Pressez GO pour que les 6 process atteignent ce breakpoint,
3 - Mettez le focus sur le rang 0 (car c'est lui qui dispose de `full_domain`),
4 - On double-clique sur `full_domain` que l'on va caster en `double[1000]`,
5 - Tools -> Visualize.

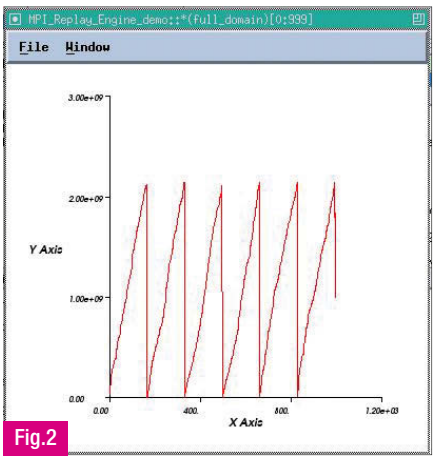


Fig.2

Le résultat escompté est le suivant : un graphique en dents de scie régulières qui tend à indiquer (visuellement au moins) que les tris partiels ont été correctement effectués puis agrégés Fig.2. Si vous êtes dans ce cas, cliquez sur le bouton Restart de TotalView (cela vous permet de redémarrer votre session de débogage sans sortir de TotalView et sans perdre les breakpoints que vous avez déjà positionnés), recommencez le procédé décrit juste avant afin d'examiner si la variable `full_domain` présente toujours la même régularité. Il y a fort à parier que vous obteniez des résultats différents d'une exécution à l'autre comme, par exemple : Fig.3 et 4.

Dans le 1er cas, on comprend que les tris partiels pour les process de rangs 2 et 4 n'ont pas fonctionné. Dans le 2nd, ce sont les process de rangs 2 et 3 qui ont failli.

A ce stade, nous avons pu confirmer au sein du débogueur le comportement non-déterministe de notre application. De plus, grâce à quelques manipulations très simples et un examen visuel sur les données, nous sommes en mesure d'affirmer que certains process n'effectuent pas correctement leur opération de tri partiel, et que ce ne sont pas les mêmes d'une exécution sur l'autre.

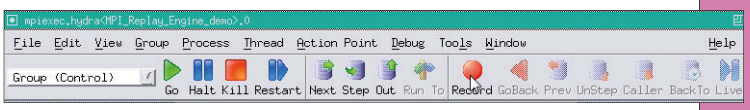
Comment venir à bout de ce casse-tête ? Il serait idéal de pouvoir enregistrer le déroulement de notre application lorsque celle-ci manifeste l'erreur. De même, on devrait de pouvoir la déboguer en marche arrière pour avoir une chance de remonter à la cause du problème. A titre de comparaison, imaginez que vous êtes en train de regarder un film DVD sur votre téléviseur ; vous pensez qu'un détail vous a échappé dans les instants précédents. Qu'allez-vous faire ? Vous allez simplement utiliser votre commande et faire un « retour arrière » jusqu'au moment qui vous intéresse, puis reprendre le cours du film. C'est précisément ce que nous allons faire ici avec TotalView, et voici comment.

Si vous avez une session TotalView ouverte, fermez-la puis démarrez-en une nouvelle :

```
> totalview -args mpiexec.hydra -n 6 ./MPI_Replay_Engine_demo
```

Avant de commencer votre session de débogage, pressez le bouton rouge **Record**. C'est cette action qui permet l'enregistrement du déroulement de notre application (qu'elle soit parallèle ou non).

Ensuite, effectuez la manipulation permettant de visualiser le contenu de



la variable `full_domain` :

1 - Positionnez un breakpoint après l'exécution de `MPI_Gather(...)` (L89 par exemple),
2 - Pressez GO pour que les 6 process atteignent ce breakpoint,
3 - Mettez le focus sur le rang 0 (car c'est lui qui dispose de `full_domain`),
4 - On double-clique sur `full_domain` que l'on va caster en `double[1000]`,

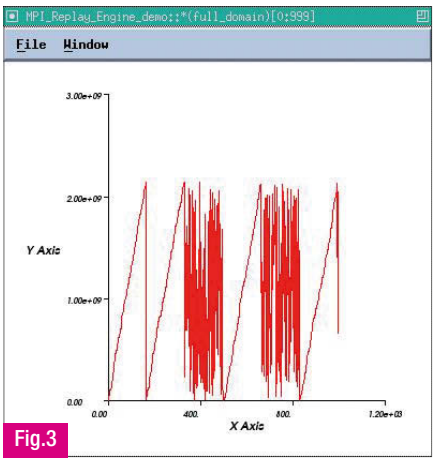


Fig.3

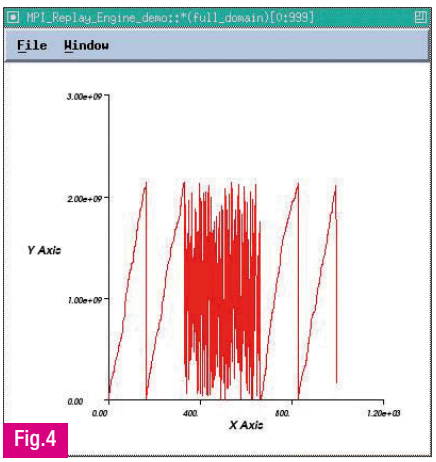


Fig.4

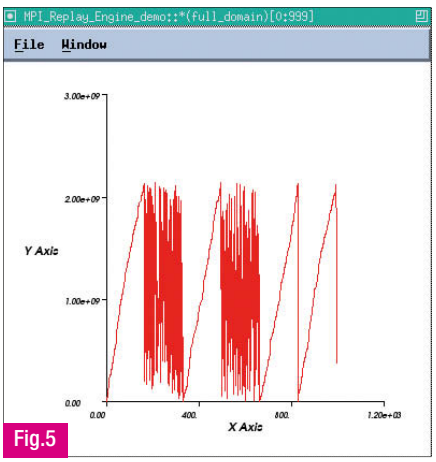
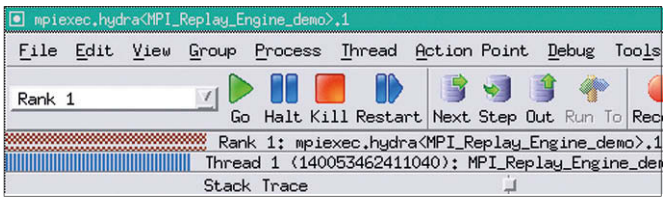


Fig.5

5 - Tools -> Visualize.
Du 1er coup, j'ai obtenu un graphe « irrégulier », exprimant très clairement que je me trouve dans un cas d'exécution « boguée ». Si vous n'avez pas cette « chance », répétez l'opération. Jusqu'à obtenir un scénario similaire (il est rare de ne pas en obtenir un au bout de 3 essais) Fig.5.
Dans mon cas, je constate que ce sont les process de rangs 1 et 3 qui posent problème. Je vais me concentrer sur le process de rang 1 et remonter le cours de son exécution, et seulement pour ce rang :



Il s'agit maintenant que je positionne en amont un point de passage pertinent à partir duquel je pourrais examiner mes variables. En l'occurrence, les tris partiels ont effectué la routine `qsort()` de la ligne [L132], routine appelée par le calcul de `local_max=getMax(…)` [L77]

```
132      qsort(array,length,sizeof(T),&myCmp<T>);
133   T lmax=array[length-1];
134   return lmax;
```

Je positionne ensuite le focus sur le rang 1 et je clique sur le bouton BackTo. Avec cette action, je demande à TotalView d'exécuter l'application depuis l'endroit où elle était arrêtée, mais en sens inverse, seulement pour le rang 1, et jusqu'à ce que celui-ci atteigne l'appel à `qsort()` que j'ai surligné. Pour faire le parallèle à nouveau avec le film DVD, imaginez que je fasse un retour-arrière jusqu'à un moment (un endroit) que j'ai précisé.

Mais TotalView m'accueille avec le message d'avertissement suivant :

C'est une très bonne nouvelle ! Pourquoi ? Parce que ce message s'interprète de la manière suivante : « j'ai remonté le cours de l'exécution du process de rang 1 jusqu'à la routine `qsort()` de la ligne L132, mais je ne l'ai jamais rencontrée dans l'historique de l'exécution ». Cela ce traduit en d'autres termes par : « pour cette exécution particulière, le process de rang 1 n'est jamais passé par la ligne L132 `qsort()` ».

```
131
132      qsort(array,length,sizeof(T),&myCmp<T>);
133   T lmax=array[length-1];
134   return lmax;
```

On peut vouloir se convaincre que le process de rang 5 passe bien, quant à lui, par la routine `qsort()`. Pour le vérifier, revenons à notre session de débogage direct en pressant le bouton Live. Mettons le focus sur le rang 5, mettons en surbrillance la ligne contenant l'instruction `qsort()` L132, et pressons le bouton BackTo. Dans ce cas, pas de message d'avertissement, mais un pointeur jaune indiquant que le rang 5 a pu remonter jusqu'à la L132, donc qu'il y est passé dans le sens direct, pour cette exécution.
Nous sommes donc parvenus à mieux cerner la cause du problème, qui se situe donc juste en amont de l'appel à `qsort()`.
Regardez la ligne 130 : en fait, `qsort()` n'est pas appelé systématiquement. Il n'est appelé que si la condition suivante est remplie : `(!myid || random() % 4)` est remplie. La voilà, la cause de notre bug non déterministe !

Offre Spéciale Fêtes

1 an de Programmez! (11 n°)

+ Clé USB Programmez!

(tous les n° depuis le n°100)

+ Accès aux archives

+ Les anciens numéros papier disponibles

TOUT **PRO**GRAMMEZ! le magazine du développeur pour

~~112,90€~~ *

79€ **

08-22-13 © esenkartal

* prix selon les stocks des numéros disponibles
** tarif valable uniquement pour la France métropolitaine

Offre spéciale valable jusqu'au 28 février 2015.

Toutes nos offres sur www.programmez.com

Oui, je profite de l'offre Spéciale Fêtes

ABONNEMENT à retourner avec votre règlement à Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex.

☐ *Abonnement 1 an au magazine + Clé USB + Accès aux archives + Les anciens numéros papier disponibles*

☐ M. ☐ Mme ☐ Mlle Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'accès aux archives et pour l'envoi d'informations relatives à votre abonnement

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez ! ☐ Je souhaite régler à réception de facture

* *Tarifs France métropolitaine*

PROG 181 *Offre limitée, valable jusqu'au 28 février 2015*

Découverte et programmation de MPI & OpenMP

Le domaine du calcul scientifique n'est pas que le domaine réservé des informaticiens. C'est un domaine multidisciplinaire où se côtoient différentes communautés. Chaque communauté n'est pas forcément experte en programmation système. Or, parallèlement à la croissance en termes de puissance, la complexité des systèmes a explosé. Afin que ces systèmes restent exploitables par des non-spécialistes, il a été nécessaire de développer des méthodes de parallélisation proposant un niveau d'abstraction compréhensible.

Michael Bacci
michael.bacci.software@gmail.com
https://www.linkedin.com/in/michaelbacci
Senior Software Engineer R&D
Expert en HPC & 3D
Michael travaille dans projets de recherche et développement pour l'industrie et institutes de recherche (universités Paris 13 et Paris 7).

Cet article est une introduction à MPI et OpenMP. Ces deux librairies intègrent des primitives pour interagir avec des systèmes distribués (MPI) et des machines à mémoire partagée (OpenMP). Elles ne sont pas opposables, elles peuvent être utilisées en combinaison dans une programmation hybride.

Tous les codes sont disponibles sur le site programmez.com

Modèles de mémoires

Dans le domaine du HPC (High Performance Computing) on distingue deux modèles de mémoire prédominants :

- Le système à mémoire partagée (Fig.1), comme par exemple OpenMP avec sa nouvelle version 4.0, qui concurrence de plus en plus toutes les librairies classiques comme la Pthread de POSIX et ses dérivées,
- Le système à mémoire distribuée (Fig.2), sur lequel se fonde le concept de MPI.

Dans le système à mémoire partagée, chaque *processus*(*) partage la mémoire principale du calculateur: les processus, au niveau hardware, peuvent lire et écrire n'importe quelle zone de mémoire (RAM), tandis qu'au niveau logiciel, c'est-à-dire à l'intérieur du programme en exécution, c'est le système d'exploitation qui gère la restriction d'accès aux régions de la mémoire.

Le système à mémoire distribuée impose une architecture où chaque processus a une zone privée de mémoire. Entre eux, les processus communiquent grâce à des messages qui sont envoyés sur le réseau. Cette communication peut être gérée, soit automatiquement dans des systèmes très complexes appelés *massively parallel computers* (MPPs), soit sur du matériel standard, par une programmation directe en utilisant des outils comme MPI. Dans ce dernier cas, on parle de cluster HPC par opposition au cluster HA (High Availability).

(*) *on entend par processus une tâche qui est exécutée dans un des composants hardware suivants: mono-processeur (cpu), multi-processeur (multi-cpu), multi-core, many-core (cpu et co-processeurs)*

1 MPI

Ce qu'est MPI

MPI c'est simplement une interface. Pour les personnes familières avec la programmation orientée objet, le concept est clair. Une interface définit une liste de routines ou fonctionnalités, avec ses paramètres en entrée et ses données en sortie, voilà tout! Le point clé c'est qu'une interface n'implémente pas ses routines. Elle les définit seulement sans donner aucun détail de construction interne. MPI cache les aspects techniques comme la mise en place de l'infrastructure de communication et la gestion des processus. Cette abstraction rend la programmation accessible aussi à des non-experts en programmation réseau et conception de logiciels. La facilité avec laquelle on peut écrire des programmes parallèles avec MPI fait que ce langage est un des plus utilisés dans le monde de l'HPC.

MPI est né de l'effort conjoint de plusieurs entreprises privées qui cherchaient, dans les années 90, un standard de communication pour l'architecture à mémoire distribuée. Il existe trois interfaces pour MPI : en langage C, C++ et Fortran 77/90. Chaque interface implémente des spécificités propres au langage concerné. Cependant, en règle générale, on y retrouve les mêmes primitives. Il y a aussi des spécifications (versions différentes), qui ont évolué au fil des années. Pour les exemples nous avons choisi l'interface en langage C.

On trouve différentes implémentations, voici les plus connues:

- "Platform MPI" de IBM [http://www-03.ibm.com/systems/platformcomputing/products/mpi/],

Fig.1

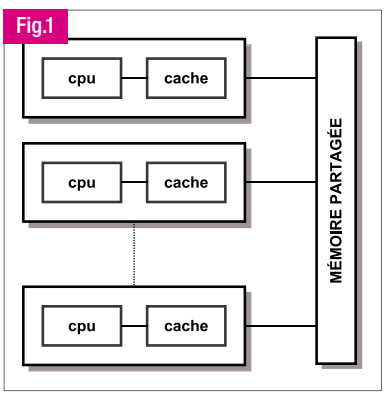


Fig.2

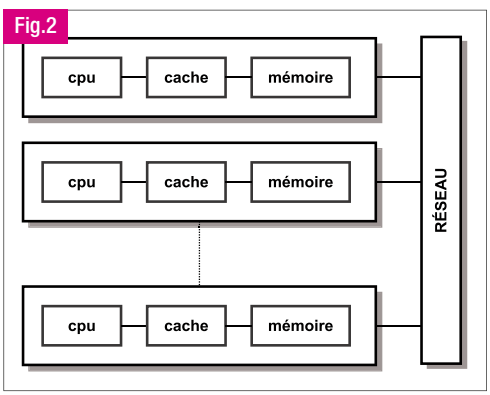
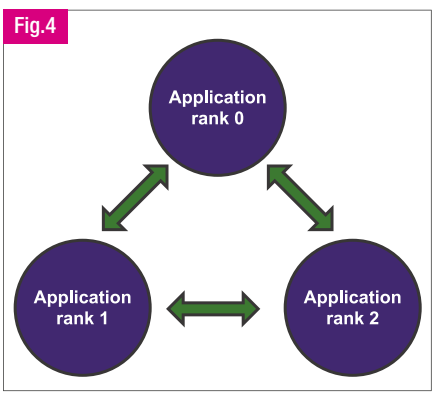


Fig.4



- "Intel MPI Library" de [https://software.intel.com/en-us/intel-mpi-library],
- "Open MPI", une implémentation open-source [http://www.open-mpi.org/].

Toutes ces librairies (et tous celles qui implémentent le standard MPI) diffèrent entre elles par rapport à :

- La vitesse d'exécution : les benchmarks (ou tests de vitesse) diffèrent beaucoup en fonction de la librairie et de l'hardware,
- La portabilité : chaque librairie a un nombre limité d'architectures hardware sur lesquelles elle peut être exécutée,
- Les outils : aide à la compilation, debug, outil de monitoring, outil de lancement de programme sur le cluster, etc.,
- Les licences : commerciale, à des fins académiques, open-source, etc.

Si vous avez envie de tester les codes d'exemples, vous pouvez télécharger le framework gratuit Open MPI à cette adresse: [http://www.open-mpi.org/software/ompi/v1.8/]

Le fonctionnement de MPI

Un programme MPI doit toujours intégrer dans le code deux informations fondamentales :

- le nombre total de processus participant à la communication, disons N,
- Un numéro unique, appelé "rank", qui identifie le processus lui-même et qui varie entre 0 et N - 1.

Chaque processus doit connaître le nombre de processus total pour savoir comment subdiviser son travail.

L'idée est celle du célèbre Jules César "Divide et Impera": diviser pour mieux régner. De la même manière que les machines communiquent entre elles grâce à leur adresse ip, les processus MPI s'identifient par leur rang (rank) pour s'échanger des messages.

Chaque processus peut être exécuté sur n'importe quel serveur ou "Noeud", sur lequel les règles suivantes sont satisfaites:

- Chaque noeud doit donner accès via le réseau en entrée et sortie (I/O) aux autres noeuds et processus qui participent à la communication,
- Chaque noeud doit installer une version de la librairie MPI compatible avec les autres nœuds.

Le point (1) peut se résoudre par un simple partage des clés publiques en utilisant le système RSA :

chaque noeud génère un couple de clés, ou bi-clés. Ce couple est composé d'une clé privée (qui sera mémorisée en local sur le noeud), et d'une clé publique (qui sera partagée avec les autres, en la mémorisant sur chaque noeud qui participe à la communication).

Pour le détail du système RSA :

[https://fr.wikipedia.org/wiki/Chiffrement_RSA].

Sur chaque noeud, on peut faire tourner un nombre quelconque de processus. Le schéma en Fig.3 montre deux noeuds : le premier avec un seul processus et le second avec deux processus.

Ces trois processus communiquent entre eux grâce à leur rang (rank) en utilisant une infrastructure de communication instanciée par MPI au dessus des réseaux physiques.

MPI présente à l'utilisateur une topologie virtuelle comme celle définie en Fig.4. Les trois processus parlent entre eux comme des unités logiques, aucune barrière ne se pose entre eux (différents systèmes d'exploitation, différentes architectures hardware et puissance de calcul). On peut ainsi concentrer l'effort de développement de l'application à la seule résolution du problème original.

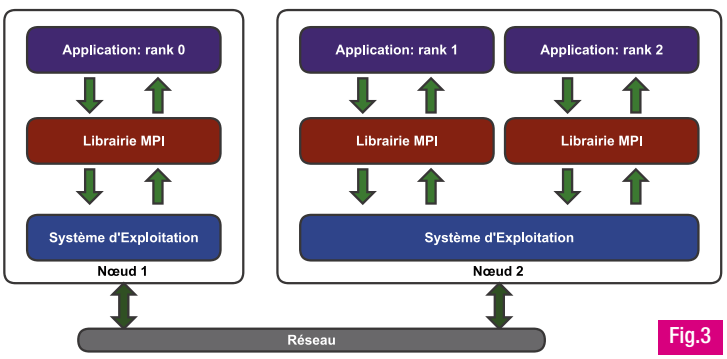


Fig.3

La communication en MPI

Pour bien maitriser MPI, il est fondamental de connaître les différentes formes de communication entre les processus.

On distingue les communications:

- Bloquante contre non bloquante,
- Point-à-point contre collective,
- Communicateurs contre groupée.

A tout message est associé un identifiant (tag), qui nous renseigne sur la nature de la communication, à savoir comment traiter la donnée échangée entre les processus.

L'idée est d'indiquer au destinataire quoi faire de la donnée reçue.

Une communication est bloquante ou synchrone lorsque les processus impliqués dans la communication
doivent attendre que le message soit totalement envoyé/reçu avant de continuer dans l'exécution du programme,
l'inverse est dite non-bloquante ou asynchrone.

La communication point-à-point implique l'interaction entre deux processus, l'un qui envoie le message, et l'autre qui le reçoit. Cette communication peut se faire de façon bloquante ou non-bloquante.

Dans la communication collective plusieurs processus prennent part à l'envoi et la réception d'un à plusieurs messages, et on y retrouve plusieurs "design pattern" ou architectures des communications.

Les "design pattern" les plus utilisés sont: scatter, gather, broadcast et reduction en Fig.5.

Les communications collectives sont uniquement bloquantes.

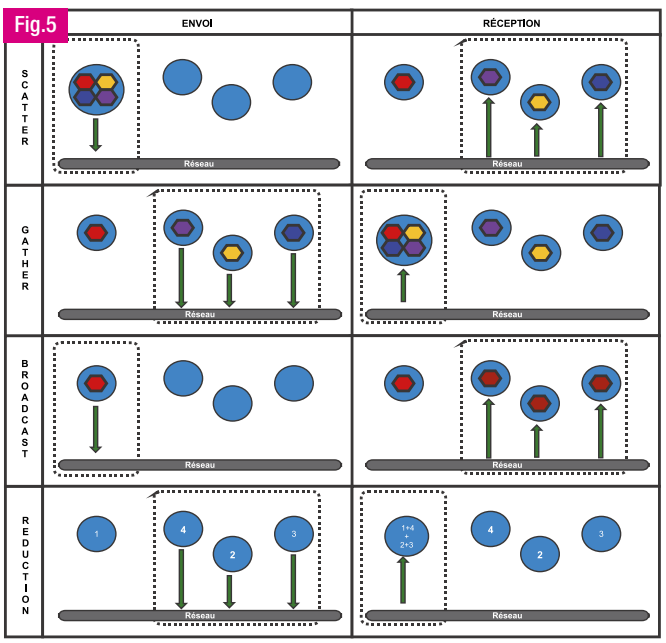
Un communicateur est un ensemble de processus.

Cet ensemble est destiné à être utilisé comme destination de communication collective.

MPI implémente par défaut un communicateur "World" (MPI_COMM_WORLD).

L'utilisateur peut définir ses propres communicateurs grâce au concept de communication groupée.

Fig.5



Structure d'un programme MPI : mpi_skeleton.c

La structure d'un programme MPI en langage C. La routine MPI_Init (&argc,&argv) initialise l'infrastructure de communication. Quand dans votre programme vous n'avez plus besoin d'utiliser aucune des routines MPI, vous devez appeler la routine MPI_Finalize(), qui sert à libérer la mémoire utilisée pour la librairie MPI, et qui, dans certains cas, met fin à la communication avec les autres processus.

Un premier programme : mpi_first_program.c

Un simple programme en MPI qui montre un schéma de fonctionnement master-slave. Dans ce schéma, un seul processus (master) gère la synchronisation des communications avec les autres processus (slave).

Analyse

Après l'usage de la routine obligatoire MPI_Init(), les routines MPI_Comm_size(), MPI_Comm_rank() et MPI_Get_processor_name() permettent au processus d'avoir les informations de base sur lui-même et sur le nombre total des processus. La variable processus_information contient donc ces informations en forme descriptive. Après l'assignation de la variable processus_information, survient une division dans la logique du programme. Le code est ainsi divisé en deux parties, l'une exécutée par le processus avec rank zéro (ou master), et l'autre, exécutée par tous les autres processus. Le code réservé au processus master effectue deux missions. La première est d'imprimer la vidéo des informations le concernant, et la deuxième, de recevoir des messages de la part des autres processus pour les imprimer à l'écran. La partie du code exécutée par les autres processus s'occupe elle-seule d'envoyer un message au noeud zéro (ou master). L'envoi et la réception des messages font partie du modèle point-à-point avec transmission bloquante. MPI_Send() est utilisé pour l'envoi du message et MPI_Recv() pour la réception. Il convient de noter que le noeud master réalise une boucle *for*, avec un index *i*, qui varie entre un et le nombre total des processus. A chaque itération, l'index *i* est égal au rang(rank) d'un des processus. En utilisant cette valeur, le noeud master utilise la routine MPI_Recv(...,i,...) pour recevoir le message de la part du processus avec rang égal à *i*. Un autre paramètre intéressant dans MPI_Send() et MPI_Recv() est MPI_COMM_WORLD. Ce paramètre définit une communication ouverte à l'ensemble des processus impliqués dans la communication.

Compilation

Pour la compilation du code, il est possible d'utiliser l'outil *mpicc* installé avec la suite des librairies d'Open MPI. Le seul code binaire produit avec *mpicc* est incapable d'exécuter plusieurs processus.
L'outil *mpirun* permet d'exécuter un code binaire compilé avec *mpicc*, sur n'importe quel noeud (machine) et avec un nombre quelconque de processus. La capture d'écran en Fig.6 montre la compilation et l'exécution du mpi_first_program.c, avec quatre processus exécutés sur la même machine. Pour exécuter le code binaire sur différentes machines, il suffit de créer un fichier avec l'adresse ip de ces machines. Il est également possible de définir le nombre exact des processus pour chaque machine.

Course de voiture : version master-slave

Avec MPI, il est possible de construire de simples jeux multiplayer. Cela est rendu possible en exploitant sa richesse en termes de modèle de communication. L'exemple mpi_game_V1.c est un simple jeu de compétition de voiture. Le jeu fonctionne ainsi : les processus non-master simulent, avec un générateur random(), le parcours d'une certaine distance avec une voiture. Tout au long du parcours, la voiture ramasse des points (ou coins). Ces informations sont envoyées au processus master. Une fois que le master a reçu les données provenant de tous les processus, ces données sont utilisées pour calculer et afficher les scores. Le jeu se termine quand tous les processus atteignent la ligne d'arrivée. Le screen-shot en Fig.7 montre un test de jeu avec dix processus, dont neuf processus (voitures) participant à la compétition et le processus master pour la synchronisation.

Analyse

Comme dans le mpi_first_program.c, on retrouve ici également le schéma master-slave et les mêmes mécanismes itératifs utilisés par le processus master pour recevoir les messages envoyés par les autres processus. Dans le programme, il y a deux types différents de messages à envoyer/recevoir, les kilomètres parcourus et les coins ramassés. Les tags *itineraire* et *bonus* sont utilisés pour distinguer les deux types de message. Les routines MPI_Send() et MPI_Recv() utilisent ces tags dans leurs paramètres. Une fois que tous les messages sont arrivés au master, celui-ci calcule le processus avec le plus de kilomètres et les coins total, afin de les afficher.

Course de voiture: version avancée

En MPI, tous les processus sont égaux. Cependant, pour la résolution de certains problèmes particuliers, le schéma master-slave se prête bien

Fig.7

```
1. bash
Michaels-iMac:MPI chopin$ mpirun -n 10 exemple_1c
Momentanément en tête le processus 7 avec 26.000000 km
Points total des joueurs: 31

Momentanément en tête le processus 8 avec 52.000000 km
Points total des joueurs: 87

Momentanément en tête le processus 1 avec 53.000000 km
Points total des joueurs: 116

Momentanément en tête le processus 1 avec 80.000000 km
Points total des joueurs: 165

Momentanément en tête le processus 1 avec 84.000000 km
Points total des joueurs: 208

Momentanément en tête le processus 2 avec 100.000000 km
Points total des joueurs: 255

Le vainqueur est le processus 2
Michaels-iMac:MPI chopin$
```

Fig.6

```
1. bash
Michaels-iMac:MPI chopin$ mpicc -std=c99 -Wall exemple_1b.c -o exemple_1b
Michaels-iMac:MPI chopin$ mpirun -n 4 exemple_1b
Je suis le processus Master. Nombre total de tâches = 4, rang = 0, Je travaille sur la machine Michaels-iMac.bouyguesbox.fr

Réception du message de la part du processus #1 ...
Message recu: Je suis un processus ESCLAVE. Nombre total de tâches = 4, rang = 1, Je travaille sur la machine Michaels-iMac.bouyguesbox.fr

Réception du message de la part du processus #2 ...
Message recu: Je suis un processus ESCLAVE. Nombre total de tâches = 4, rang = 2, Je travaille sur la machine Michaels-iMac.bouyguesbox.fr

Réception du message de la part du processus #3 ...
Message recu: Je suis un processus ESCLAVE. Nombre total de tâches = 4, rang = 3, Je travaille sur la machine Michaels-iMac.bouyguesbox.fr
Michaels-iMac:MPI chopin$
```

à la solution. Dans le programme d'exemple **mpi_game_V1.c** le noeud master ne participe pas à la course de voiture.
mpi_game_V2.c utilise le modèle de communication collective. Grâce à cette architecture de communication, le noeud master participe à la compétition tout en gardant un code extrêmement compact et bien plus élégant.

Analyse

Les différences avec la version du **mpi_game_V1.c** sont multiples. Le concept de tag ne sert plus. Chaque couple de routines MPI_Send/MPI_Recv est remplacé avec une seule routine MPI_Reduce. Non seulement MPI_Reduce réalise l'envoi et la réception du message, mais il calcule automatiquement les scores! Pour mieux comprendre MPI_Reduce voici un tableau avec ses paramètres.

int MPI_Reduce(	Description des paramètres
void *sendbuf,	adresse du buffer en envoi
void *recvbuf,	adresse du buffer en réception
int count,	nombre d'éléments à envoyer
MPI_Datatype datatype,	types d'éléments contenus dans le buffer
MPI_Op op,	opération de réduction (max,min,sum,prod...)
int root,	le rank qui recevra les données
MPI_Comm comm);	type de communicateur

Pour calculer le processus avec le plus de kilomètres, chaque processus envoie son rang et le parcours réalisé à travers la variable de structure *in*. MPI_Reduce calcule et mémorise dans la variable *out* le rang et le kilomètre du processus gagnant la course. C'est le paramètre MPI_MAXLOC qui permet de calculer un maximum local sur la structure *_competition_*. Cette structure est de forme MPI_FLOAT_INT, c'est-à-dire qu'elle est composée par une première variable de type *float* et d'une seconde variable de type *int*. Pour le calcul des points total des joueurs, chaque processus attribue une valeur à la variable coins. L'opération de réduction MPI_SUM calcule la somme de tous les *coins* envoyés par le processus. Le résultat est mémorisé dans la variable *tot_coins*.

Note: Les vrais jeux multiplayers sont réalisés sur des modèles de communication asynchrone. Le code ci-dessus peut être transformé en version asynchrone en utilisant la programmation hybride et la routine de communication asynchrone.

MPI en 8 point clés

MPI est composé de 8 sections bien séparées, qui sont :

Type	Exemples
Gestion environnement	MPI_Init, MPI_Finalize
Communication point-à-point	MPI_Send,MPI_Recv, MPI_Isend, MPI_Irecv
Communication collective	MPI_Bcast,MPI_Scatter,MPI_Gather,MPI_Reduce
Communication de groupe	MPI_Comm_group,MPI_Group_incl,MPI_Group_rank
Communicateur	MPI_Comm_size,MPI_Comm_create,MPI_Comm_rank
Data dérivée	MPI_Type_commit,MPI_Type_indexed,MPI_Type_free
Topologie virtuelle	MPI_Cart_create, MPI_Cart_coords,MPI_Cart_shift
Routine d'utilité	MPI_Pack, MPI_Unpack

2 OPENMP

OpenMP est une API pour le contrôle du parallélisme sur le modèle de mémoire partagée. Elle est définie par trois composants:

- Une librairie à runtime,
- Des directives (#pragma omp) pour le compilateur,
- Un ensemble de variables définies dans l'environnement d'exécution.

Lorsque l'on cherche à implémenter un code parallèle en utilisant les PThreads de POSIX, la classe Thread de Java, ou bien la librairie std::thread de C++11, il faut utiliser des méthodes et structures de données propres à chacune de ces librairies. OpenMP fonctionne de manière différente dans la mesure où il intègre la notion de directive de compilation couramment appelée *pragma*.
Grâce à cette notion de pragma, OpenMP communique directement avec le compilateur pour l'informer sur les optimisations à faire au niveau des accès de mémoire, des gestions de threads et lock/unlock des ressources. OpenMP a des origines communes avec MPI. OpenMP (qui signifie Open Multi-Processing) est né dans les années 90, grâce à l'effort conjoint d'entreprises privées et d'institutions de recherche publiques. Cet effort commun était devenu une nécessité pour proposer une interface unifiée à la multitude d'architectures à mémoire partagée.
Les objectifs d'OpenMP sont :

- Simplicité de programmation,
- Fonctionnement sur un grand nombre d'architecture et plateformes différentes,
- Limitation du nombre de directives pour transformer un programme séquentiel en parallèle,
- Support des langages: Fortran(77/90/95), C et C++

Le site officiel se trouve à l'adresse suivante : [http://openmp.org/wp/]
Pour savoir si votre compilateur est capable d'interpréter les directives d'OpenMP, voici l'adresse avec la liste des compilateurs compatibles : [http://openmp.org/wp/openmp-compilers/]
La programmation en multithreading peut être réalisée de différentes façons. Il existe beaucoup de librairies qui implémentent ce paradigme de programmation.
Cependant, seul OpenMP a une telle facilité d'utilisation qui aide le développeur à éviter les erreurs dans ce genre de domaine de programmation.
OpenMP permet une programmation *non-intrusive*: avec une simple directive, on peut rendre une boucle itérative de séquentielle à parallèle, même sans rien y connaitre en programmation parallèle!

Modèle de programmation

OpenMP offre un modèle de programmation explicite.
Le développeur a le contrôle total de chaque aspect de la programmation. Pour l'implémentation de son modèle, OpenMP utilise la technique appelée *fork-join* décrite en **Fig.8.**
Le fonctionnement est le suivant :

- Chaque programme en OpenMP démarre avec un seul thread, appelé "master thread",
- Le *master thread* continue son exécution jusqu'à ce qu'il rencontre une région parallèle,
- Dans le *Fork* (ou début de la région parallèle), le master thread crée un ensemble de threads, dont le nombre peut être défini de manière explicite dans le code, ou bien en utilisant la variable d'environnement OMP_NUM_THREADS,
- Dans la région parallèle, chaque thread a accès à une zone de mémoire privée et/ou partagée, selon les directives données au compilateur par le programmeur de l'application,

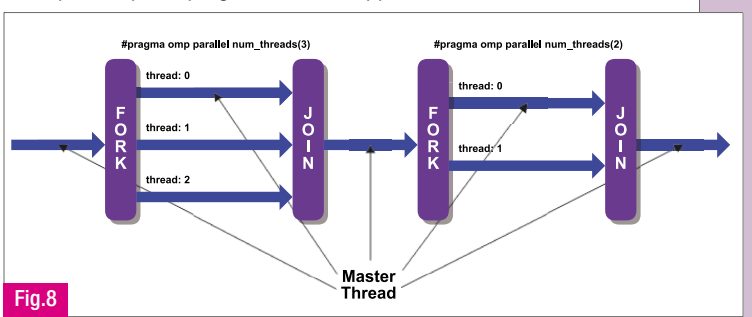


Fig.8

- Le *Join* (ou fin de la région parallèle) synchronise les threads, les termine et reprend l'exécution du master-thread.

OpenMP en action

Pour bien comprendre la puissance d'OpenMP, on peut faire une simple expérience. On va créer un programme séquentiel, et on va le paralléliser en utilisant MPI (que maintenant nous connaissons) et ensuite avec OpenMP.
serial.c montre la version séquentielle de ce simple programme.
mpi_sum.c montre la version parallèle en MPI, et, pour la version en OpenMP : openmp_sum.
openmp_sum montre la facilité d'utilisation d'OpenMP.
Avec une seule directive on a réussi à informer le compilateur de la section du code qui doit être parallélisée.
En argument, la variable *somme* qui doit être partagée entre les différents threads, est spécifiée.

OpenMP en détails

Tous les exemples qui suivront peuvent être compilés avec l'option "-fopenmp" sur votre compilateur gcc et clang. Pour les exemples utilisant les fonctions run-time, il faut inclure la librairie <omp.h>.
Nous montrerons des exemples en langage C. Cependant ces exemples seront parfaitement compatibles en C++.
Il faut faire particulièrement attention lorsqu'on utilise en C++ la STL (ou standard library) avec les directives pragma d'OpenMP. La STL de C++ n'est pas thread-safe, il faut donc utiliser des mécanismes de blocage (lock) pour éviter des accès concurrents à la même ressource.

La syntaxe

Toutes les directives d'OpenMP commencent avec *#pragma omp* suivies des paramètres et attributs.

Paramètre: parallel

Définir une région parallèle a pour effet de déclencher une séquence fork-join suivant le modèle décrit ci-dessus.
Voici des exemples :

```
#pragma omp parallel
{
 /* tout le code présent ici est exécuté par chaque threads */
 printf("Je suis le thread %d, Nombre total des threads %d\n",
   omp_get_thread_num(), omp_get_num_threads());
}
```

```
1. bash
Michaels-iMac:MPI chopin$ gcc-mp-4.8 -fopenmp -Wall
-std=c99 exemple_parallel_for.c -o parallel_for
Michaels-iMac:MPI chopin$ export OMP_NUM_THREADS=4
Michaels-iMac:MPI chopin$ ./parallel_for
Je suis le thread 0, je travaille sur l'index 0
Je suis le thread 2, je travaille sur l'index 6
Je suis le thread 1, je travaille sur l'index 3
Je suis le thread 3, je travaille sur l'index 9
Je suis le thread 0, je travaille sur l'index 1
Je suis le thread 2, je travaille sur l'index 7
Je suis le thread 1, je travaille sur l'index 4
Je suis le thread 3, je travaille sur l'index 10
Je suis le thread 0, je travaille sur l'index 2
Je suis le thread 2, je travaille sur l'index 8
Je suis le thread 1, je travaille sur l'index 5
Je suis le thread 3, je travaille sur l'index 11
Michaels-iMac:MPI chopin$
```

Fig.9

Le nombre de threads dans la zone parallèle varie en fonction de l'architecture sur laquelle le programme est en exécution. On peut aussi définir une variable d'environnement de la manière suivante :

```
export OMP_NUM_THREADS=4
```

Cette variable d'environnement a pour effet de forcer l'exécution à 4 threads. Nous pouvons aussi le faire directement dans le code avec l'attribut *num_threads*, de la manière suivante :

```
#pragma omp parallel num_threads(4)
{
   ...
}
```

Attribut: for

Cet attribut divise la boucle itérative en plusieurs threads. Chacun de ces threads partage une portion des données.

```
#pragma omp parallel for
for (int i=0; i < 12; ++i)
{
 printf("Je suis le thread %d, je travaille sur l'index %d\n",
 omp_get_thread_num(), i);
}
```

Supposons que nous ayons 4 threads, on obtiendra un résultat comme en Fig.9.
Les résultats ne sont pas ordonnés. Les actions sont exécutées par le programme de manière concurrente.
Aucune directive n'a été donnée quant à l'organisation ou à la synchronisation de ces actions.
Or, ces actions se réalisent en parallèle sur plusieurs unités de calcul. Le résultat n'est donc pas déterministe, c'est à dire que les affichages vont se réaliser de manière aléatoire.
Si on ordonne les résultats en fonction de l'index de la boucle *for*, on obtient un schéma comme en Fig.10.
Ce schéma nous montre comment OpenMP décide de partager la boucle itérative entre les différents threads.
Les directives pragma peuvent être enchainées les unes dans les autres. Dans ce cas, l'attribut *for* de la directive *parallel* devient une directive valable. Le code précédent, est transformable dans la façon suivante :

```
int donne_totale = 12;
#pragma omp parallel
{
 int current_thread = omp_get_thread_num();
```

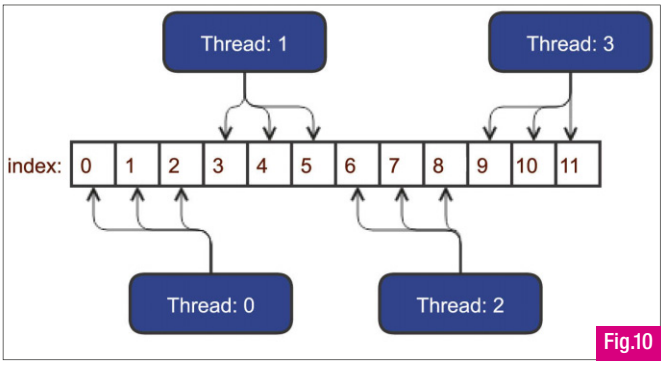


Fig.10

```
#pragma omp for
  for (int i = 0; i < donne_totale; ++i) {
    printf("Je suis le thread %d, je travaille sur l'index %d\n",
      current_thread, i);
   }
}
```

L'usage de la directive *#pragma omp for* est appelée approche *fine-grain*. Cette approche est non-intrusive et permet une écriture rapide du code. L'organisation et la synchronisation des threads sont laissées au compilateur. Cette stratégie a par conséquence des coûts supplémentaires en termes de nombre d'instructions pour la cpu.
Il existe une deuxième approche, dite *corse-grain*, qui élimine ces coûts supplémentaires. Dans le *corse-grain*, le partage des tâches est totalement défini par le développeur.
Le code *fine-grain* peut être ainsi transformé en *corse-grain* :

```
int donne_totale = 12;
#pragma omp parallel
{
  int current_thread = omp_get_thread_num();
  int tot_threads = omp_get_num_threads();
  int start =  (current_thread * donne_totale) / tot_threads;
  int end   = ((current_thread + 1) * donne_totale) / tot_threads;

  for (int i = start; i < end; ++i)
    printf("Je suis le thread %d, je travaille sur l'index %d\n",
  current_thread, i);
}
```

Scheduling

Dans l'approche *fine-grain*, des attributs sont disponibles pour contrôler la portion des données partagées entre les threads. Les boucles itératives sont divisées en parties appelées chunk. Le chunk est un nombre entier qui définit le nombre d'itérations disponibles pour chaque thread. Ces attributs sont :

- *static*: c'est l'attribut par défaut, chaque thread choisit la portion de la boucle *for* où il travaillera et calculera son chunk
- *dynamic*: une fois le chunk calculé, chaque thread aura une portion aléatoire dans la boucle itérative
- *guided*: comme l'attribut *dynamic*, mais avec la valeur du chunk qui diminue de manière proportionnelle à chaque itération.

Le scheduling est réalisé avec le paramètre *schedule*.
Voici un exemple :

```
#pragma omp parallel for schedule(static, 3) num_threads(2)
  for (int i=0; i < 12; ++i)
```

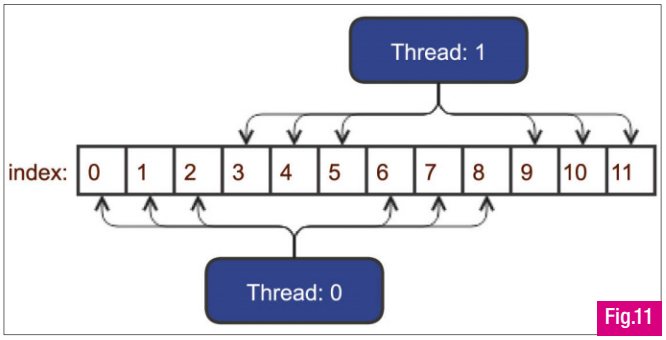


Fig.11

```
    printf("Je suis le thread %d, je travaille sur l'index
%d\n", omp_get_thread_num(), i);
```

Ce morceau de code est représenté de manière schématique en Fig.11.

Les attributs *private* et *shared*

Dans une section parallèle, il est possible de définir, pour chaque variable, un accès privé ou partagé entre les threads. Par défaut, toutes les variables définies avant une section parallèle sont partagées entre tous les threads.
Les variables définies dans le bloc de code parallèle sont privées.
L'attribut *private(var1,var2,...)* crée une copie des variables énumérées, dans la zone de mémoire privée de chaque thread.
L'attribut *shared(var1,var2,...)* partage toutes les variables énumérées entre les threads. Voici un exemple :

```
int i;
double somme = 0;
const double pi = 3.14;

#pragma omp parallel private(i) shared(somme)
{

  /**
    * variable privée: chaque thread a une copie
    * de somme_partielle et i dans sa zone de mémoire privée
    */
  double somme_partielle = 0;

#pragma omp for
  for (i = 0; i < 1024; ++i) {
    somme_partielle += pi * i;
  }

  #pragma omp atomic
  somme += somme_partielle;
}
```

La directive *atomic*, dans le code ci-dessus, prévient les accès concurrents à la variable somme. Le paragraphe suivant explique comment prévenir les accès concurrents.

Sécurité des accès mémoire

Le partage des ressources constitue l'aspect le plus complexe de la programmation parallèle. Une variable, un fichier ou une connexion au réseau sont autant d'exemples de ressources à partager entre les threads. Afin de partager correctement les ressources, il est nécessaire de réaliser une synchronisation des threads. Lock, mutex et sémaphore sont les mécanismes réalisant cette synchronisation.
Un lock est la porte d'accès à une section du code source. L'accès est réservé à un seul thread à la fois.
Le thread qui possède le lock ne le partage pas avec les autres threads. Le mutex est un lock, mais qui peut être défini au niveau du système d'exploitation. L'écriture concurrente sur un fichier, entre processus différents, est gérée par le système d'exploitation grâce à la création d'un mutex sur le fichier. Un sémaphore est comme un mutex, mais capable de donner accès à une ressource à plusieurs threads en même temps. La directive *critical* restreint l'accès à une zone parallèle, un seul thread est admis à la fois. L'ordre des accès à une zone définie *critical* suit l'ordre d'arrivée des threads à cette zone. Plusieurs instructions sont

admises dans cette section. La directive *atomic*, comme la directive *critical*, garantissent l'accès exclusif à des ressources.
Cependant, une seule instruction est admise dans la directive *atomic*. Cette restriction a l'avantage d'optimiser la compilation.
L'optimisation est faite pour les seules opérations de multiplication, d'addition, de division et de soustraction d'une variable numérique. Dans ces cas, le compilateur est capable de réduire le temps de blocage (lock) des threads en attendant de mettre à jour la variable.
Voici un exemple avec les directives *critical* et *atomic* dans la même portion de code parallèle.

```
char message[100];
int somme = 0;
int position_message = 0;

#pragma omp parallel num_threads(3)
{
 #pragma omp critical
 {
   sprintf(&message[position_message],
     "Je suis le thread %d\n", omp_get_thread_num());
   position_message = strlen(message);
  }

 #pragma omp atomic
 somme += omp_get_thread_num();
}

printf("%s\n", message);
printf("somme = %d == (0 + 1 + 2)\n", somme);
```

Le programme peut générer un résultat comme celui-ci :

```
Je suis le thread 1
Je suis le thread 0
Je suis le thread 2

somme = 3 == (0 + 1 + 2)
```

L'univers d'OpenMP

OpenMP est composé de nombreuses directives et attributs. Cependant, celles montrées ci-dessus sont les directives de base pour créer votre programme parallèle.

OpenMP dans la vraie vie

OpenMP est souvent utilisé pour créer des versions parallèles d'une même librairie. Bien que OpenMP soit souvent utilisé dans le domaine scientifique, on le retrouve aussi dans des produits
pour le grand public. On va prendre comme exemple la librairie de manipulation d'images ImageMagick [http://www.imagemagick.org/].
La librairie existe en version single-thread et multi-threads. La version multi-thread utilise OpenMP. Les développeurs d'ImageMagick se sont appuyés sur les facilités proposées par OpenMP.
Nous allons illustrer cela avec un petit exemple.
Supposons une fonction qui applique à chaque pixel une correction alpha, voici un pseudo-code séquentiel.

```
void AppliqueAlpha(Image *image, const float alpha) {

 for (int i = 0; i < image->num_totale_de_pixel; ++i) {
   image->pixel[i] *= alpha;
 }
}
```

Voici la version parallèle du même code :

```
void AppliqueAlpha(Image *image, const float alpha) {

#ifdef OPENMP_ENABLED
#pragma omp parallel for
#endif
 for (int i = 0; i < image->num_totale_de_pixel; ++i) {
   image->pixel[i] *= alpha;
 }
}
```

La variable de préprocesseur OPENMP_ENABLED est définie au moment de la compilation de la librairie.
La variable est passée au compilateur de la façon suivante : "-D OPENMP_ENABLED". En utilisant cette astuce, vous pouvez créer des versions parallèles de votre code, pour créer des librairies multi-plateformes (car OpenMp est multi-plateforme) et multi-thread.

LA PROGRAMMATION HYBRIDE EN QUELQUES MOTS

Chaque modèle d'architecture parallèle cherche à exploiter au mieux les ressources disponibles. Pour cela ont été construits des paradigmes de programmation qui leur conviennent le mieux. Afin d'exploiter la totalité des systèmes hétérogènes existants, la programmation hybride crée un mélange des différentes techniques de programmation et librairie existantes. Le mélange le plus utilisé est le binôme composé par MPI et OpenMP. Avec MPI on peut exploiter toute la puissance d'un réseau de calculateurs, et avec OpenMP réussir à utiliser tous les *cores* dont chaque unité d'élaboration est fournie. Un des aspects les plus importants de ce binôme est le gain de mémoire. Grâce à la mémoire partagée d'OpenMP, la duplication des données entre les différents noeuds est considérablement réduite. En conséquence, la librairie interne d'MPI utilise beaucoup moins de mémoire pour son fonctionnement. Pour rendre le **mpi_sum.c** hybride, il faut seulement ajouter deux étapes : Informer MPI que le programme exécutera des threads, Rajouter avec OpenMP la clause de parallélisation dans la partie de somme des données. On obtient le code hybride : **hybrid.c**
La routine MPI_Init() a été remplacée par MPI_Init_Thread().
Ce changement est nécessaire pour définir la modalité des interactions entre les threads et les routines MPI.
Il existe quatre modalités d'interactions possibles :

- MPI_THREAD_SINGLE: seul un thread est supporté,
- MPI_THREAD_FUNNELED: le processus peut être multi-thread, mais seul le master-thread a le droit d'utiliser les routines MPI,
- MPI_THREAD_SERIALIZED: processus multi-thread, tous les processus ont le droit d'utiliser les routines MPI mais seulement une à la fois,
- MPI_THREAD_MULTIPLE : processus multi-thread, tous les processus ont le droit d'utiliser les routines MPI sans restrictions.

La routine MPI_Init_Thread() envoie une requête au système de la machine pour avoir la configuration demandée. Avec MPI_Query_thread(), il est possible de savoir si la configuration demandée a été correctement réalisée.

Programmation multithread en Java à l'heure de Java 8

Dès sa création, la plateforme Java a proposé aux développeurs un excellent support pour la programmation multithread. Avec le temps, les méthodes pour implémenter la concurrence en Java ont bien évolué avec pour objectif de simplifier au maximum le travail du développeur. Le but étant bien entendu de l'abstraire de la complexité inhérente à la programmation concurrente afin de réduire les risques d'erreurs. Dans cet article, nous nous proposons de faire le tour des solutions pour mettre en œuvre la programmation multithread en Java à l'heure de Java 8.

Sylvain Saurel
Ingénieur d'Etudes Java / JEE
sylvain.saurel@gmail.com

Le passage à l'an 2000 aura marqué un tournant dans le monde informatique. Finie la progression continue de la puissance des processeurs et place désormais à l'ajout de cœurs afin d'augmenter les performances. Les processeurs multi-cœurs sont désormais la norme quel que soit le type d'appareil, et la tendance va en s'accentuant.

Face à cette orientation, les développeurs doivent monter en compétence sur la programmation multithread et apprendre à adresser les problématiques induites, afin d'en tirer le meilleur parti au sein de leurs programmes. Dans ce contexte, le développeur Java est plutôt chanceux puisque la plateforme met à sa disposition tout un tas d'APIs pour la programmation concurrente. Dans ce qui suit, nous ferons un tour d'horizon de ces outils tout en introduisant d'abord les concepts de base du multithreading en Java.

Bases du multithreading

Au sein de la plateforme Java, les threads sont représentés par la classe java.lang.Thread et leur exécution est réalisée au sein d'instances d'objets java.lang.Runnable. Les développeurs peuvent créer un grand nombre de threads au sein de leurs programmes et la machine virtuelle se charge de la répartition sur les différents cœurs à l'exécution. Si le nombre de threads lancés dépasse le nombre de cœurs à disposition, les threads supplémentaires sont partagés sur les cœurs.

La première difficulté inhérente au multithreading est liée à la coordination des actions entre threads. Elle tient au fait que la machine virtuelle Java est libre d'ordonner les opérations comme bon lui semble au sein du code tant que cela demeure consistant du point de vue du programme. Ainsi, si des additions utilisent des variables différentes, le compilateur ou la JVM peuvent les exécuter dans l'ordre inverse de ce qui est spécifié du moment que le programme n'utilise pas la somme avant que les opérations soient terminées. Cette flexibilité confère à la plateforme le pouvoir d'améliorer les performances de Java mais la consistance n'est garantie qu'à l'intérieur d'un même thread.

Il est bon de noter que le hardware peut lui aussi être source de problèmes avec les threads étant donné que les systèmes modernes utilisent différents niveaux de cache mémoire et que ces derniers ne sont pas vus de manière identique par les différents cœurs du système. Ceci implique donc qu'un changement de valeur en mémoire réalisé au sein d'un cœur pourrait ne pas être immédiatement visible par les autres cœurs.

Ces problématiques obligent les développeurs à contrôler explicitement les interactions entre threads lorsqu'ils travaillent avec des données partagées. Pour réaliser ces opérations de contrôle, Java met à disposition des mots-clés dédiés au sein du langage permettant d'établir l'ordre dans lequel les données seront vues par chaque thread employé. L'opération de base pour un thread est d'utiliser le mot-clé synchronized pour accéder à un objet. Lorsqu'un thread synchronise un objet, il obtient un accès exclusif à un lock unique sur cet objet. Si un autre thread possède déjà un lock sur ce même objet, le thread voulant l'obtenir va devoir attendre ou bloquer son exécution jusqu'à ce qu'il soit libéré. Ainsi, quand un thread exécute du code au sein d'un bloc synchronized, la JVM garantit que ce thread peut voir le résultat de tout ce qui a été réalisé par les autres threads ayant précédemment obtenu ce même lock jusqu'au moment de sa libération. Ceci résout la problématique de ré-ordonnancement des opérations et celle de cache mémoire. On pourrait ainsi comparer le contenu d'un bloc synchronized à une zone de stabilité au sein d'un programme où les threads peuvent chacun à leur tour s'exécuter, interagir et partager des données de manière sûre.

Le second mot-clé à connaître est volatile. Positionné sur une variable, il va séparer la garantie offerte par synchronized en 2 parties distinctes. Si un thread écrit dans une variable marquée par volatile, toutes les valeurs précédentes sont effacées pour cette variable. Si un thread lit une variable volatile, il verra la valeur correspondant à tout ce qui a été écrit par les autres threads. Accéder à une telle variable au sein d'un thread fournit une forme de garantie mémoire équivalente au code d'un bloc synchronized à ceci près que la lecture et l'écriture d'une variable volatile sont non bloquantes.

Rajout d'un niveau d'abstraction

Un grand nombre d'applications Java multithreads sont développées en recourant uniquement aux mécanismes de synchronisation pour la gestion des interactions entre threads. Néanmoins, cela requiert une maîtrise certaine de la complexité que cela induit. De plus, gérer de nombreux threads et locks conduit bien souvent à des situations de deadlocks. Un deadlock étant une situation où au moins 2 threads attendent chacun que l'autre libère ses locks afin de poursuivre son exécution. Dès lors, les architectes de la plateforme ont souhaité ajouter un niveau d'abstraction pour la mise en place de la concurrence afin d'abstraire le développeur de cette complexité en proposant des outils répondant simplement à de nombreux cas d'utilisations communs.

L'API java.util.concurrent a donc été introduite incluant des implémentations de collections supportant les accès concurrents, des classes wrappers pour opérations atomiques et des primitives pour la synchronisation. Les différentes classes ayant été modélisées afin de proposer des accès non bloquants ce qui évite les problèmes de deadlocks tout en améliorant l'efficacité du multithreading. Bien qu'elles facilitent les interactions entre threads, ces classes souffrent encore d'un certain nombre de défauts.

Cette même API a donc été étendue avec un nouveau niveau d'abstraction autorisant une approche mieux découplée du multithreading : l'inter-

face Future<T> ainsi que les interfaces Executor et ExecutorService. Future<T> permet de manipuler une valeur de type T en partant du principe qu'elle n'est généralement pas disponible immédiatement après la création de l'objet Future. La valeur est le résultat d'une opération asynchrone pouvant être exécutée de manière concurrente. Le thread recevant un Future peut appeler des méthodes pour vérifier si la valeur est disponible, attendre qu'elle le soit, la récupérer ou annuler l'opération si la valeur n'est plus attendue. Différentes implémentations de Futures sont fournies afin de modéliser différents types d'opérations.

L'interface Executor est une abstraction pour l'exécution de tâches bien souvent indissociable de sa sous-interface ExecutorService. Cette dernière proposant des méthodes pour produire les objets Future et gérer leur exécution de manière transparente. Enfin, un mot sur l'implémentation ThreadPoolExecutor qui vient simplifier la gestion d'un pool de threads. Les threads étant gourmands en ressources, cela va du bon sens que de les réutiliser plutôt que de les allouer à nouveau.

Place à la pratique

Les fondamentaux du multithreading en Java exposés, nous pouvons passer à la pratique. Compte tenu de la taille de l'article, cet exemple sera basé sur un problème mathématique, certes moins complexe que des problèmes du monde réel avec des interactions externes mais dont la résolution permettra de montrer la plupart des techniques utilisables en Java à l'heure actuelle. Nous nous intéressons donc à la distance de Levenshtein qui mesure la similarité entre 2 chaînes de caractères. Sur un grand nombre de mots à comparer, il s'avère payant de lancer le calcul sur une machine multi-cœurs en tirant partie du multithreading. La stratégie va donc être de découper le nombre de mots à comparer en sous-ensembles et d'effectuer le traitement de chacun de ces sous-ensembles comme des tâches séparées. La tâche de calcul de la distance est déléguée à un objet DistanceTask implémentant l'interface Callable :

```
public class DistanceTask implements Callable<DistancePair> {
  private final String targetText;
  private final int startOffset;
  private final int compareCount;

  // … Constructeur

  private int editDistance(String word, int[] v0, int[] v1) {
    // ...
  }

  @Override
  public DistancePair call() throws Exception {
    int[] v0 = new int[targetText.length() + 1];
    int[] v1 = new int[targetText.length() + 1];
    int bestIndex = -1;
    int bestDistance = Integer.MAX_VALUE;
    boolean single = false;

    for (int i = 0; i < compareCount; i++) {
      int distance = editDistance(knownWords[i + startOffset], v0, v1);

      if (bestDistance > distance) {
        bestDistance = distance;
        bestIndex = i + startOffset;
        single = true;
      } else if (bestDistance == distance) {
        single = false;
      }
    }

    return single ? new DistancePair(bestDistance, knownWords
[bestIndex]) : new DistancePair(bestDistance);
  }
}
```

A l'exécution de la tâche, la méthode call est appelée et c'est en son sein qu'est effectué le travail de comparaison pour le sous-ensemble démarrant à startOffset et de longueur compareCount. Ici, on remarque le recours à une classe valeur DistancePair associant une distance à un mot donné. Elle possède une méthode statique best permettant de comparer 2 instances de DistancePair en renvoyant la meilleure, c'est-à-dire celle ayant la distance la plus petite avec le mot d'origine. Enfin, elle propose une méthode statique worstMatch() renvoyant une implémentation la plus mauvaise possible pour un mot vide. La valeur de la distance étant égale à Integer.MAX_VALUE dans ce cas-là. Nous pouvons maintenant implémenter le calcul parallélisé de la distance :

```
public class ThreadPoolDistance {
  private final ExecutorService threadPool;
  private final String[] knownWords;
 private final int blockSize;

 public ThreadPoolDistance(String[] words, int block) {
   threadPool = Executors.newFixedThreadPool(Runtime.getRuntime().
availableProcessors());
   knownWords = words;
   blockSize = block;
 }

 @Override
 public DistancePair bestMatch(String target) {
   List<DistanceTask> tasks = new ArrayList<DistanceTask>();
   int size = 0;

   // Construction des sous-ensembles
   for (int base = 0; base < knownWords.length; base += size) {
     size = Math.min(blockSize, knownWords.length - base);
     tasks.add(new DistanceTask(target, base, size));
   }

   DistancePair best;

   try {
     // Invocation sur l'Executor
     List<Future<DistancePair>> results = threadPool.invokeAll(tasks);

     // Calcul du meilleur résultat, attente du retour de chaque Future
     best = DistancePair.worstMatch();

      for (Future<DistancePair> future: results) {
       DistancePair result = future.get();
       best = DistancePair.best(best, result);
     }

   } catch (InterruptedException e) {
     throw new RuntimeException(e);
   } catch (ExecutionException e) {
```

```
      throw new RuntimeException(e);
    }

    return best;
  }
}
```

Au sein de la méthode bestMatch, on construit une liste de tâches avec leurs sous-ensembles de mots à traiter. On invoque ensuite cette liste via la méthode invokeAll de l'instance d'ExecutorService créée à la construction de ThreadPoolDistance. On remarque la méthode factory newFixedThreadPool qui prend en entrée le nombre de cœurs disponibles sur la machine hôte. La méthode invokeAll renvoie en sortie une liste de Futures sur laquelle nous bouclons afin d'attendre le résultat de chacune des tâches exécutées. Au final, la variable best contient l'instance de DistancePair représentant le mot ayant la plus faible distance avec le mot passé en entrée.

Fork-Join

Avec Java 7, une nouvelle implémentation d'ExecutorService a été introduite au travers de la classe ForkJoinPool. Modélisée pour gérer efficacement ce qui peut être divisé en sous-tâches, la nouvelle API propose RecursiveAction pour une tâche ne devant produire aucun résultat et la classe RecursiveTask<T> lorsqu'un résultat de type T doit être produit. En sus, la méthode Join va faciliter la consolidation des résultats obtenus comme nous pouvons le voir dans le code suivant :

```
public class ForkJoinDistance {
  private ForkJoinPool threadPool = new ForkJoinPool();
  private final String[] knownWords;
  private final int blockSize;

  // … Constructeur

  public DistancePair bestMatch(String target) {
    return threadPool.invoke(new DistanceTask(target, 0, known
Words.length, knownWords));
  }

  public class DistanceTask extends RecursiveTask<DistancePair> {
    private final String compareText;
    private final int startOffset;
    private final int compareCount;
    private final String[] matchWords;

    public DistanceTask(String from, int offset, int count,
String[] words) {
      compareText = from;
      startOffset = offset;
      compareCount = count;
      matchWords = words;
    }

    protected int minimum(int a, int b, int c) {
      return Math.min(Math.min(a, b), c);
    }

    private int editDistance(int index, int[] v0, int[] v1) {
      // ...
    }

    @Override
    protected DistancePair compute() {
      if (compareCount > blockSize) {
        // division en 2 sous-tâches
        int half = compareCount / 2;
        DistanceTask t1 = new DistanceTask(compareText, start
Offset, half, matchWords);
        t1.fork();
        DistanceTask t2 = new DistanceTask(compareText, start
Offset + half, compareCount - half, matchWords);
        DistancePair p2 = t2.compute();
        return DistancePair.best(p2, t1.join());
      }

      // comparaison
      int[] v0 = new int[compareText.length() + 1];
      int[] v1 = new int[compareText.length() + 1];
      int bestIndex = -1;
      int bestDistance = Integer.MAX_VALUE;
      boolean single = false;

      for (int i = 0; i < compareCount; i++) {
        int distance = editDistance(i + startOffset, v0, v1);

        if (bestDistance > distance) {
          bestDistance = distance;
          bestIndex = i + startOffset;
          single = true;
        } else if (bestDistance == distance) {
          single = false;
        }
      }

      return single ? new DistancePair(bestDistance, knownWords
[bestIndex]) : new DistancePair(bestDistance);
    }
  }
}
```

La classe DistanceTask se charge de séparer la tâche de comparaison courante en 2 sous-tâches tant que le nombre de mots à traiter est supérieur à ce qu'une tâche peut traiter. Une fois le nombre de mots à traiter inférieur à la taille d'un bloc de traitement, on effectue la comparaison au sein de la tâche de la même manière que vu précédemment. Au sein du bloc conditionnel de séparation en 2 sous-tâches, on notera le lancement de la première tâche avec l'appel à fork et l'appel à compute sur la seconde sous-tâche. Le résultat des 2 tâches étant consolidé via join.

Retour vers les Futures

Les Futures proposés par la plateforme avant Java 8 étaient plutôt restreints en termes de fonctionnalités puisque supportant uniquement 2 usages : vérifier la terminaison de l'exécution d'un Future ou attendre la fin de son exécution. Pour étendre les fonctionnalités de cette première implémentation, Java 8 propose la classe CompletableFuture<T>, implémentant l'interface CompletionStage<T>, et héritant de Future<T>. Au niveau conceptuel, CompletionStage représente une étape dans une opération potentiellement asynchrone. Le contrat d'interface définit de nombreuses méthodes pour interagir avec ses implémentations.
Pour séparer la tâche de comparaison en sous-tâches, nous créons cette

fois-ci une classe ChunkDistanceChecker qui propose les méthodes editDistance et bestDistance ainsi qu'une méthode statique de création des sous-ensembles nommée buildCheckers. Le programme de traitement concurrent a alors l'allure suivante :

```
public class CompletableFutureDistance {
  private final List<ChunkDistanceChecker> chunkCheckers;
  private final int blockSize;

  public CompletableFutureDistance0(String[] words, int block) {
    blockSize = block;
    chunkCheckers = ChunkDistanceChecker.buildCheckers(words, block);
  }

  @Override
  public DistancePair bestMatch(String target) {
    List<CompletableFuture<DistancePair>> futures = new ArrayList<>();

    for (ChunkDistanceChecker checker: chunkCheckers) {
      CompletableFuture<DistancePair> future = CompletableFuture.
supplyAsync(() -> checker.bestDistance(target));
      futures.add(future);
    }

    DistancePair best = DistancePair.worstMatch();

    for (CompletableFuture<DistancePair> future: futures) {
      best = DistancePair.best(best, future.join());
    }

    return best;
  }
}
```

On voit clairement les apports des Lambdas de Java 8 en termes de concision ainsi que la facilité d'emploi des CompletableFutures en contexte concurrent. La méthode supplyAsync prend une instance de l'interface fonctionnelle Supply<T> en entrée et retourne une instance de CompletableFuture<T> en mettant en queue le supplier pour une exécution asynchrone. Ce travail est réalisé au sein de la boucle parcourant les sous-ensembles créés. Enfin, la seconde boucle attend que chaque CompletableFuture termine son exécution pour garder le meilleur résultat. Bien évidemment, et comme précédemment, le travail des Futures sera déjà terminé au moment de l'exécution de cette boucle for du fait de leur exécution en mode asynchrone.

Streams

Autre nouveauté majeure de Java 8, les Streams peuvent travailler de concert avec les Lambdas expressions. Ils peuvent être chaînés et proposent des opérations de filtrage et de mapping. Le code précédent peut ainsi être écrit de la sorte avec des Streams au niveau de la méthode bestMatch :

```
 public DistancePair bestMatch(String target) {
  return chunkCheckers.stream()
         .map(checker -> CompletableFuture.supplyAsync(() -
> checker.bestDistance(target)))
         .collect(Collectors.toList())
         .stream()
         .map(future -> future.join())
         .reduce(DistancePair.worstMatch(), (a, b) -> Distance
Pair.best(a, b));
 }
```

On créé tout d'abord un Stream à partir de la liste de ChunkDistanceChecker puis on applique un mapping avec la méthode CompletableFuture.supplyAsync pour obtenir une exécution asynchrone de la méthode bestDistance. On collecte le résultat au sein d'une liste de Futures avec la méthode collect.
On ouvre à nouveau un Stream pour boucler sur les CompletableFutures et on applique une Lambda expression afin d'attendre le résultat d'exécution de chacun d'entre eux.
Enfin, il ne reste plus qu'à effectuer une opération de réduction via un appel à reduce en appliquant la méthode DistancePair.best() entre le résultat précédent et le meilleur jusque-là.

Streams parallèles

Bien que concise et claire pour quelqu'un d'habitué aux Lambdas de Java 8, la solution mêlant Streams, Lambdas et CompletableFutures peut paraître légèrement indigeste et, pire encore, source d'erreurs. Fort heureusement, Java 8 propose une implémentation parallèle des Streams. Là où un Stream séquentiel est créé via un appel à stream(), un Stream traité en parallèle par la machine virtuelle est ouvert via parallelStream().
Une fois cet appel réalisé, la machine virtuelle garantit que toutes les opérations suivantes appliquées au Stream seront réalisées automatiquement au sein de threads séparés avant de collecter les résultats lorsque nécessaire.
Notre implémentation du calcul de la distance de Levenshtein combinant Streams et Lambdas expressions en mode multithread devient tout à coup beaucoup plus lisible :

```
public DistancePair bestMatch(String target) {
  return chunkCheckers.parallelStream()
         .map(checker -> checker.bestDistance(target))
         .reduce(DistancePair.worstMatch(), (a, b) -> Distance
Pair.best(a, b));
}
```

On reconnaît les appels aux méthodes map et reduce vues dans l'exemple de code avec les Streams séquentiels mais il n'est plus nécessaire de manipuler explicitement des instances de CompletableFutures et d'attendre leur résultat. Tout est parallélisé automatiquement ce qui offre un gain de productivité non négligeable pour les développeurs tout en réduisant au maximum la complexité inhérente au multithreading.

Conclusion

Bénéficiant jusqu'alors d'un excellent support pour la programmation concurrente, la plateforme Java s'est vue doter de nouveaux outils avec la sortie de sa version 8. Comme on a pu le voir tout au long de cet article, l'apparition des CompletableFutures mais plus encore celle des Lambdas expressions et des Streams, modifie clairement la mise en œuvre de la programmation concurrente en Java.
Le niveau d'abstraction supplémentaire apporté par ces nouveautés devant permettre au développeur de se concentrer uniquement sur le code métier tout en relevant le défi du multithreading avec un maximum de productivité et un minimum de risques d'erreurs.
Attention toutefois à la montée en compétences nécessaire pour bien maîtriser les Lambdas expressions et les Streams afin ne pas s'y perdre devant tant de concision.

Scala : mixer objet et fonctionnel

Scala a 10 ans. Son père, Martin Odersky, peut être fier. Après avoir travaillé sur le compilateur Java pour y introduire, entre autres, les génériques, Martin se sentait trop vite limité par le rythme d'évolution de Java, langage éprouvé qui tire sa richesse de sa communauté importante et de la Java Virtual Machine (JVM). Scala est donc né avec un objectif en tête : réunir la programmation orientée objet et fonctionnelle. C'est un langage alternatif sur la JVM. Il vient avec un compilateur qui transforme le code source en bytecode permettant deux avantages : utiliser une machine virtuelle éprouvée et profiter de toutes les bibliothèques Java.

Bastien **Bonnet**,
consultant chez Xebia

Xebia

Xavier **Bucchiotty**,
consultant chez Xebia

Scala contient de nombreuses facilités syntaxiques permettant de réduire grandement la verbosité du code pour se concentrer sur la valeur ajoutée du programme.
À l'instar de Java, Scala est un langage typé statiquement. Il se différencie ainsi d'autres langages sur la JVM comme Groovy ou Clojure. Même après la sortie de Java 8 et ses nouvelles fonctionnalités issues du monde fonctionnel, Scala reste très utile. Nous allons détailler les points qui différencient ces deux langages, tant sur l'approche objet que l'approche fonctionnelle.

Scala, un Java musclé

Scala permet une programmation orientée objet. À ce titre, vous pouvez organiser votre code en classes et interfaces. Malgré les différences entre le langage Java et Scala, leur syntaxe reste proche l'une de l'autre. Avec l'objectif d'avoir un langage expressif, plusieurs mots-clés ont été supprimés par rapport à Java. On entend parfois que le code Scala est compliqué à lire ; la suite de cet article va vous surprendre.

Inférence de type

Scala possède un bon moteur d'inférence de type. C'est-à-dire que le compilateur est capable de déduire le type des variables en fonction de leurs utilisations.

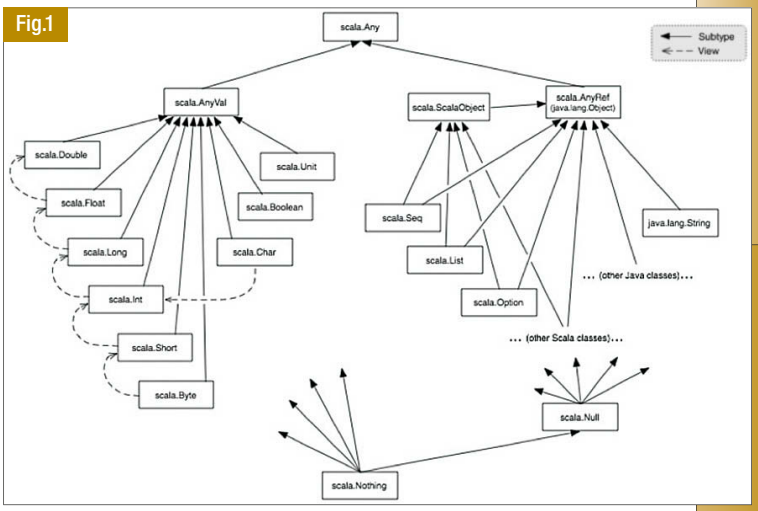
```
//Java
Integer a = 0;

//Scala
val a = 0
```

Le compilateur déduit de l'affectation que 'a' est de type '*Int*'. Il faut par contre toujours donner explicitement le type des variables pour les membres de classes et les paramètres de méthodes. Par contre, il est conseillé de donner le type de sortie d'une méthode pour la lisibilité : le compilateur peut s'en charger.

No public admitted

Avec le mot-clé **public**, le langage Java déclare un élément visible de l'extérieur. Très souvent répété, Scala a décidé d'en faire la visibilité par défaut et donc de le faire disparaître de son langage. Dans un langage qui prône l'immuabilité, le danger est moindre (cf le paragraphe « Immuabilité »). En effet, un objet immuable n'a pas besoin de respecter le principe d'encapsulation, ainsi personne ne peut modifier son état. Plusieurs études

Fig.1



La hiérarchie des types en Scala (source : http://docs.scala-lang.org/).

montrent qu'un développeur passe plus de temps à lire du code qu'à en écrire. Moins il y a de bruit dans le code, plus il est simple de le comprendre. Le mot-clé **public** ne représente peut être pas grand chose, mais répété toutes les 10 lignes, il freine la compréhension, sans amener d'informations utiles ! Les mots-clés **private** et **protected** existent toujours et gardent le même rôle qu'avec Java. Scala va plus loin avec **private[this]**. Avec cette visibilité, deux instances de la même classe ne peuvent pas accéder à cet élément, ce qui est impossible à faire en Java.

Pas de static

Pour rappel, en Java, le mot-clé **static** définit le fait que le membre auquel il s'applique est partagé par toutes les instances de la classe concernée. Ce mot-clé ne relève pas de l'objet, car il n'existe pas d'héritage possible. Scala enrichit donc son approche objet et le supprime.
Pour le remplacer, Scala possède un nouveau mot-clé : **object**. Utilisé à la place de **class**, il permet de créer un singleton.
La réelle différence entre **static** et **object** est néanmoins fondamentale. Un **object** est, comme son nom l'indique, un objet possédant les mêmes possibilités qu'une classe, comme le polymorphisme d'héritage. Le raisonnement est plus simple et la fonctionnalité puissante.
Concrètement, pour gérer les valeurs ou méthodes communes à toutes les instances d'une classe, il vous suffit de créer une classe et un objet portant le même nom dans le même fichier. C'est la notion d'Objet Compagnon. Ces deux entités liées par leur nom peuvent accéder à leurs attributs privés. Cette technique plus élégante permet de séparer concrètement les variables et méthodes de classes et d'instances.

Arborescence des types

En Scala, l'arborescence des types est bien plus simple à appréhender car il n'existe pas de cas particulier comme en Java avec les types prim-

itifs (*int, double*) et les autres classes. L'illustration 1 montre la hiérarchie des types en Scala. Le supertype de tous les types est Any. Deux sous-classes en héritent : AnyVal et AnyRef. Les types « simples » (value types en Anglais) sont des sous-classes de AnyVal : Boolean, Int, Char, etc. Les autres types sont des sous-classes de AnyRef.
Null est un sous-type de tous les AnyRef et Nothing le sous-type de Any.

Pas de type primitif ou d'opérateur

Le fait qu'il n'existe pas de type primitif dans le langage Scala renforce l'orientation objet : tout est objet et méthode. Par exemple, 1 + 1 revient à écrire 1.+(1). Il existe une méthode nommée + sur la classe *Int* : plus besoin d'autoboxing, plus de problème de hiérarchie. Concernant les performances, le compilateur se charge d'inliner les méthodes des types primitifs. Il n'y a pas d'overhead au runtime, seulement à la compilation.

Un seul constructeur

Avec le langage Scala, toute classe possède un unique constructeur primaire et éventuellement des constructeurs auxiliaires. Voici un exemple de classe simple avec un constructeur primaire :

```
class Personne (val nom: String, val anneeDeNaissance: Int)
```

Les paramètres du constructeur primaire sont déclarés immédiatement après le nom de la classe, entre parenthèses. Ces paramètres sont transformés en champs qui seront initialisés avec les paramètres effectifs. La déclaration équivalente en Java serait la suivante :

```
//Java
public class Personne {
  private String nom;
  private int anneeDeNaissance;

  public Personne(String nom, int anneeDeNaissance) {
    this.nom = nom;
    this.anneeDeNaissance = anneeDeNaissance;
  }

  public String nom() { return this.nom; }
  public int anneeDeNaissance() { return this.anneeDeNaissance; }
}
```

On peut observer au passage la concision avec laquelle Scala exprime les mêmes fonctionnalités. Si l'on désire ajouter des constructeurs, on peut définir des constructeurs auxilliaires, définis par l'identifiant **this**, qui doivent obligatoirement commencer par un appel à un constructeur précédemment défini.

Case class

En Java, on trouve souvent des classes simples contenant des données et quelques méthodes utilitaires telles que equals, hashCode et toString. En Scala, les *case classes* nous évitent d'écrire beaucoup de code :

```
// Scala
case class Employé(nom: String, métier: String, salaire: Int)
```

Cette seule ligne permet de générer une classe avec ses attributs, accesseurs et méthodes utilitaires : equals, hashCode, toString, copy, ainsi que des méthodes permettant le pattern matching (cf. paragraphe "Pattern matching").

Trait

Les traits Scala permettent de définir des comportements, de la même manière que les interfaces en Java, mais avec la possibilité de définir des implémentations par défaut, seulement disponibles depuis Java 8. Cependant, les traits offrent en plus la possibilité de gérer les états et de restreindre les types pouvant les implémenter. Une classe Scala peut implémenter n'importe quel nombre de traits.
Un trait se définit de la manière suivante :

```
trait A {
def hello: String = "hi A"
 }
```

Et s'utilise de la manière de cette façon :

```
class B extends A
```

Comme il est possible pour une classe d'implémenter plusieurs traits, chacun ayant possiblement des fonctions concrètes, que se passe-t-il si une classe implémente deux traits définissant chacun la même méthode ? Scala a trouvé une solution simple à cela : le compilateur le détecte et produit une erreur si l'on ne redéfinit pas la fonction en question :

```
trait A {
 def hello: String = "hi A"
}
trait B {
 def hello: String = "hi B"
}
class C extends A with B // Erreur de compilation
```

Pour éviter cette erreur, redéfinissez explicitement la fonction en question, en utilisant le mot-clé **override** :

```
class C extends A with B {
    override def hello: String = super[B].hello
}
```

Généricité et variance

Comme en Java, il est possible d'avoir des types génériques en Scala. Ils permettent de définir des classes et fonctions qui fonctionnent avec plusieurs types, comme par exemple :

```
class Paire[A, B](val premier: A, val second: B)
```

À l'utilisation, cela donne :

```
val p = new Paire("John", 2)
```

Il est également possible d'ajouter des contraintes sur les types. Par exemple, si l'on souhaite pouvoir comparer les valeurs dans notre paire, il est possible d'imposer que les valeurs soient comparables, c'est-à-dire soient une sous-classe de Comparable :

```
class Paire[A <: Comparable[A]](val premier: A, val second: A)
```

Programmer avec les types génériques s'avère vite périlleux: combien de développeurs Java ont été forcés de faire du cast pour régler certaines API? Scala résout ce problème en donnant accès à une fonctionnalité avancée, la variance. Cela permet de définir la relation d'héritage qu'il y a entre un type et son paramètre polymorphique. Voyons cela avec un exemple :

```
val employés: List[Employé] = List(employé_1)
//en Java, la ligne suivante ne compile pas
val personnes: List[Personne] = employés
```

Programmation fonctionnelle

Nous avons vu qu'il était possible avec Scala de construire des programmes orientés objets robustes et élégants. Avec l'approche fonction-

nelle, Scala ouvre de nouvelles perspectives aux développeurs. Pour un développeur provenant de Java, la courbe d'apprentissage était jusqu'ici souple, elle s'accentue avec cette nouvelle face du langage. Il faut apprendre à résoudre certains problèmes différemment et oublier certains réflexes. La majeure partie des critiques concernant Scala ne sont souvent pas liées au langage lui-même mais à la programmation fonctionnelle.

Le style impératif se caractérise par une approche *bottom-up*. Il faut exprimer ce que la machine va exécuter, au risque de noyer et perdre de vue le but de plus haut niveau. L'approche fonctionnelle adopte une approche *top-bottom*, se définissant comme un style déclaratif. Le résultat attendu est décrit, laissant l'implémentation technique aux briques de plus bas niveaux.

Prenons un problème de géométrie simple. Il faut trouver l'ensemble des triangles dont le périmètre est de 24 cm.

Voici une solution en style impératif, suivie d'une approche fonctionnelle en Scala :

```
//Java
List<Triangle> rightTriangles = new ArrayList<>();

for(int i = 1; i < 10; i++){
  for(int j = 1; j <= i; j++ ){
    for(int k = 1; k <= j; k ++){
      if( i*i == (j*j + k*k) && i + j + k == 24){
        rightTriangles.add(new Triangle(i,j,k));
      }
    }
  }
}

//Scala
val rightTriangles = for {
  i <- 1 to 10
  j <- 1 to i
  k <- 1 to j if i*i == (j*j + k*k) && i + j + k == 24
} yield(i,j,k)
```

Alors qu'en Java, il faut faire des boucles imbriquées, créer plusieurs variables temporaires et une structure de données, Scala propose une solution plus concise et néanmoins expressive.

À l'exécution, les deux solutions sont équivalentes. Cependant, la seconde méthode se contente de décrire le problème, masquant au lecteur les détails techniques.

Convaincu par cette solution ? Découvrons ensemble les clés de la programmation fonctionnelle en Scala.

Function as first-class-citizen (Code as data)

On dit d'un langage qu'il est fonctionnel si la fonction est une entité première. C'est le cas pour plusieurs langages comme Haskell, Clojure ou Scala. Il est possible de déclarer des fonctions anonymes, de passer des fonctions en paramètres et retours d'autres fonctions. Toute fonction peut être stockée dans une variable. Le code devient alors une donnée, au même titre qu'un entier ou une chaîne de caractères. Son invocation ne dépend de rien d'autres que de ses arguments d'entrée. Elle peut être combinée avec d'autres fonctions pour former des fonctions plus complexes. La composition de fonction est le mécanisme de base de construction de programmes complexes. Cela induit une manière de penser différente, proche de l'algèbre en mathématiques.

En Scala, les méthodes d'une classe sont aussi considérées comme des fonctions. Contrairement à Java8, il n'y a pas d'opérateur différent pour accéder à la représentation en fonction d'une méthode. Les deux déclarations suivantes s'utilisent donc de manière équivalente :

```
object MonObjet {
  def maMethode(input: String): String = …
  val maFonction = (input: String) => …
}

List("a").map(MonObjet.maMethode)
// est équivalent à
List("a").map(MonObjet.maFonction)
```

Immuabilité

On dit qu'une structure est immuable s'il est impossible de modifier son contenu. Toutes mises à jour s'effectuent par copie totale ou partielle de la source. Cela permet d'éviter les effets de bord. Scala promeut cette approche. C'est notamment visible dans l'API Collection. La hiérarchie est disponible dans les deux modes, muable ou non. La version immuable est celle importée par défaut. De prime abord, de nombreux développeurs pensent que les structures immuables sont moins performantes, ce qui n'est pas si évident. En Java, pour une collection, l'implémentation *ArrayList* est principalement utilisée. On peut facilement la parcourir, y ajouter un élément au début, à la fin ou modifier un élément au milieu. Un développeur qui découvre Scala pense pouvoir faire la même chose avec *List* mais est rapidement déçu.

L'utilisation de structure immuable sera moins versatile qu'une ArrayList. Pour s'en rendre compte, il suffit de parcourir la documentation de l'API Collection de Scala :

	head	tail	apply
immutable			
List	C	C	L
mutable			
ArrayBuffer	C	L	C

C	The operation takes (fast) constant time.
L	The operation is linear, that is it takes time proportional to the collection size.
head	Selecting the first element of the sequence.
tail	Producing a new sequence that consists of all elements except the first one.
apply	Indexing.

source : http://www.scala-lang.org/docu/files/collections-api/collections_40.html

Si ces deux implémentations permettent d'accéder en temps constant au premier élément de la collection, leurs comportements diffèrent pour les autres opérations. Ainsi le choix de l'implémentation par le programmeur doit être réfléchi en fonction de son utilisation. Cela peut apparaître comme étant un défaut mais cette prise de conscience est intéressante. Elle nous pousse à nous poser les bonnes questions, le plus tôt possible.

Le second avantage de l'immuabilité est sa facilité à raisonner dans un environnement concurrent ou distribué. Même si un objet est modifié par plusieurs threads, chacun aura sa copie qui sera intègre. Le problème de résolution reste présent. Le programmeur doit choisir sa stratégie.

En Java, vouloir utiliser des structures immuables est punitif. Il faut ajouter le mot-clé "final", les API sont rarement pensées dans ce sens et lancent bien souvent des exceptions pour signaler une opération interdite.

En Scala, les API sont très souvent sans opération de mutation. Définir une variable muable ou immuable ne prend pas plus de caractères dans le code. On utilise "var" pour une variable muable, "val" pour une immuable. De plus, certains IDE colorent les variables muables pour signaler un danger potentiel.

Pattern matching

Le pattern matching est un outil puissant. À première vue, il s'agit d'un simple switch ou d'une suite de *if* chaînés. En Scala, le pattern matching peut être utilisé de trois façons différentes :

- égalité à des constantes,
- vérification du type,
- déconstruction.

L'égalité à des constantes s'apparente à une structure de type « switch » en Java.
En Java, si vous testez la classe d'un objet, vous devez ensuite faire un "downcast" pour récuperer l'objet dans cette implémentation. En Scala, cela est fait automatiquement.
La dernière utilisation s'appelle la déconstruction. Prenons l'exemple du parsing d'un email depuis une chaîne de caractères. En Scala, le pattern matching fonctionne très bien avec une expression rationnelle :

```
val Email: Regexp = """(.*)@(.*)\.(.*)""".r

"info@xebia.fr" match{
  case Email(name,domain,extension) => println("your name is $name")
  case _ => println("sorry")
}
```

Chaque élément du pattern est passé séparément sous forme de paramètre.
Si la variable ne correspond pas à un des cas précisés, le programme évalue la ligne suivante. Il n'y a rien de magique derrière tout cela. Cette déconstruction d'email n'est rien d'autre qu'une fonction qui a pour argument une chaîne de caractères qui retourne un triplet si l'expression est vérifiée.
La déconstruction permet, en une seule ligne, de créer une condition sur la donnée en entrée et de la transformer. "::" représente la fonction de construction d'une liste. *Nil* étant la liste vide, *1 :: Nil* correspond à la même chose que *List(1)*.

```
List(1,2,3) == 1 :: 2 :: 3 :: Nil
```

Il est aussi possible d'utiliser la même syntaxe pour la déconstruction :

```
List (1,2,3) match{ case a :: b :: c :: Nil => …}
```

For comprehension

Le "for comprehension" est l'un des sucres syntaxiques les plus utilisés en Scala. Il vient à la base des raisonnements mathématiques. Oubliez l'instruction *for* qui permet de faire des boucles en Java, celui-ci aura un sens légèrement différent. Prenons l'exemple d'une itération simple :

```
val list = List(1,2,3)

val result = for{
  x <- list
}yield x
```

Première chose à noter, l'instruction *for* retourne une valeur. Il est donc possible d'affecter ce résultat à une variable. Il n'y a pas de compteur ni de comparaison avec la taille de la liste.
La dernière différence que l'on peut noter est l'apparition du mot-clé *yield*. Cette instruction contient l'expression qui sera appelée pour chaque ligne trouvée de la liste. Au final, cette ligne peut se lire : "Le résultat est égal à la liste qui : Pour tout x contenu dans list, retourne x"
Cette instruction est bien plus puissante qu'en Java. Il est par exemple possible d'imbriquer des boucles :

```
val list_1 = List(1,2,3)
val list_2 = List(2,3)

val result = for{
  x:Int <- list_1
  y:Int <- list_2
} yield x + y
> result = List(3, 4, 4, 5, 5, 6)
```

L'itération sur list_2 est faite pour chaque valeur de x dans list_1, d'où un résultat avec 3*2=6 éléments.
Il est aussi possible d'utiliser des conditions de filtres avec cette instruction :

```
val list_1 = List(1,2,3)
val list_2 = List(2,3)

val result = for{
  x <- list_1 if x % 2 == 0
  y <- list_2
} yield x + y
> result = List(4, 5)
```

« Pour tout x contenu dans list_1 pour qui x est pair, et tout y contenu dans list_2, retourne x + y »
Encore plus fort ? Cette instruction fonctionne non seulement pour toutes les collections, mais aussi pour bien d'autres types de données, comme les options et même les futures.
Prenons l'exemple avec deux futures :

```
def fetchPrice:Future[Int] = ???
def fetchQuantity : Future[Int] = ???

val total : Future[Int] = for{
  price <- fetchPrice
  quantity <- fetchQuantity
}yield price * quantity
```

price et *quantity* sont deux entiers issus des futures fetchPrice et fetchQuantity. Total est à son tour une variable de type Future[Int]. Ce *for* enregistre un traitement qui sera exécuté lorsque les deux requêtes auront retourné une valeur. Ce résultat sera alors stocké dans *total*. Tout est asynchrone. Comment est-ce possible ? Voici le code qui est réécrit par le compilateur.

```
val total = fetchPrice.flatMap{ price =>
    fetchQuantity.map{ quantity =>
      price * quantity
    }
  }
}
```

Chaque itération est remplacée par l'appel à une méthode *flatMap*, le contenu de yield se trouve dans un map. Enfin, si nous avions utilisé des filtres, ils auraient été appliqués avec la méthode "filter". Dans Scala, cela fonctionne sur tous les types qui ont une méthode flatMap, map et filter où résident les détails d'implémentation du comportement du 'for'.

Outillage & Écosystème

Une richesse, la Java Virtual Machine

Un des atouts majeurs de Scala est d'être exécuté sur la Java Virtual Machine. En effet, les programmes bénéficient des vingt dernières

années de R&D dans le domaine de la gestion de la mémoire et de la concurrence. Un programme écrit en Scala peut être packagé dans un fichier .jar, .war, .ear ou même un script. Il est possible alors de les intégrer dans un batch, des applications sur Android, une application Web ou dans un script d'exploitation.
Étant sur la JVM, Scala bénéficie de l'inépuisable liste des bibliothèques Java, couvrant ainsi un large panel de besoins. Les usines logicielles Java sont aussi compatibles avec Scala.

Rétro-compatibilité des binaires

Scala est un langage qui comporte beaucoup de fonctionnalités. Pour y parvenir, de nombreuses modifications ont été apportées au travers des années, au prix de certaines incompatibilités entre versions. À la différence de Java qui garantit une rétrocompatibilité dans ses changements de versions, la communauté autour de Scala a fait le choix de s'autoriser quelques changements.
Aujourd'hui, il existe deux compilateurs. Le premier est maintenu par la société Typesafe, le second par la communauté Typelevel. Nous voilà confrontés au même type de séparation entre les distributions Debian et Ubuntu. Cela n'entame en rien leur popularité, bien au contraire !
Typesafe fournit une version stable du langage. Typelevel assure que sa version du langage sera plus riche que celle fournie par Typesafe, tout en restant compatible. Ainsi, les "early-adopters" de ce compilateur pourront profiter des dernières fonctionnalités et faire vivre le langage.

Scala REPL & SBT

Dans l'idée de faciliter le travail quotidien des développeurs, Scala apporte dans sa distribution un REPL (Read-Eval-Print-Loop). En lançant dans un terminal la commande "scala", un environnement d'évaluation est lancé. Il permet au programmeur d'essayer le langage, sans avoir besoin de créer un projet, un test ou une classe Main. Le terminal évalue ligne par ligne ce qu'on lui fournit, offrant un retour visuel immédiat. C'est idéal pour la découverte, le prototypage et la création de scripts.
SBT, pour Simple Build Tool, est un outil de build pour la JVM. Il est écrit en Scala et est la plateforme privilégiée pour les projets en Scala. Sa DSL de configuration peut laisser un néophyte perplexe. Son avantage évident est d'être un script compilé. Cela permet d'être guidé par les types lors de son utilisation.
Dans le même esprit, SBT est un REPL. C'est d'ailleurs pour cela que la page d'accueil du site de SBT affiche maintenant fièrement "Interactive Build Tool". Une fois lancé, il est possible de lancer des tâches classiques comme "compile", "clean" et "test". Les temps de compilation de Scala étant connus pour être importants, le fait de disposer d'un REPL est un gain considérable en phase de développement. Tous les caches du compilateur restent chauds, assurant une compilation incrémentale extrêmement efficace. C'est aussi idéal pour le TDD, en ajoutant un simple '~' devant une commande, elle est exécutée à chaque changement de fichiers sources. Donc utiliser '~test' permet de lancer toutes la série de tests à chaque changement de fichiers.

Une nouvelle gamme de bibliothèques pointues

Dans les domaines des applications distribuées, de systèmes hautement concurrents et de hautes performances, de plus en plus de frameworks et d'applications viennent de la communauté Scala.

Gatling

On peut citer Gatling, un outil de stress d'applications permettant de simuler une forte activité sur les serveurs. L'outil fournit ensuite des rapports sur la qualité de réponses des services. Gatling repose sur la richesse du langage pour fournir une DSL décrivant facilement des scénarios complexes de charge.

Apache Spark

Apache Spark est un outil distribué de traitement et d'agrégation de données. Écrit en Scala, il offre la possibilité de lire en streaming, filtrer et transformer de très gros volumes de données, avec une syntaxe claire et efficace. Il s'interface aussi parfaitement avec Apache Hadoop.

Akka

Akka est un autre framework qui rend populaire Scala. Son utilisation du paradigme acteur permet de bâtir sereinement des systèmes distribués, concurrents et hautement disponibles. Un acteur est à la frontière entre un objet et une fonction. Il communique en échangeant des messages immuables avec d'autres acteurs. Cela correspond parfaitement à la philosophie du langage.

Ressources

La communauté autour de Scala est de plus en plus vivante et déborde d'envies de partager son intérêt pour cet écosystème. Paris, Nice et Lyon accueillent régulièrement un Scala User Group. Les sujets sont variés, du niveau débutant au plus expert, chacun peut y trouver un intérêt. Scala possède aussi sa propre conférence à Paris, Scala.IO. Cette année a eu lieu la seconde édition. Il s'agit de deux jours de conférence sur Scala et plus généralement des langages fonctionnels, des problématiques des systèmes distribués et hautement scalables.
Scaladays est une conférence annuelle organisée par Typesafe. Les sessions sont visibles dès aujourd'hui sur le site Parleys.
Le site de cours en ligne gratuit héberge deux cours en Scala, Functionnal Programming Principale in Scala et Principles of Reactive Programming. Les deux cours sont menés par Typesafe et l'École Fédérale Polytechnique de Lausannes avec Martin Odersky en personne. Scalacourses est un autre site Web permettant de se former et se renseigner sur ce langage et sa communauté.

Conclusion

Dans les tâches quotidiennes, Scala est un langage peu verbeux, qui permet de construire simplement des systèmes robustes et de fournir des réponses simples à des problématiques complexes. Cependant, Scala implique maîtrise, rigueur et patience.
Ce langage est à conseiller à une population de développeurs qui a envie d'apprendre, de s'améliorer et qui aime se challenger. Un programmeur provenant du monde Java connaîtra dans un premier temps un réel plaisir à programmer avec Scala. Tout ce qui était verbeux ou perfectible en Java devient immédiatement possible et rapide. Mais, il se heurtera rapidement à une nouvelle façon de penser avec l'approche fonctionnelle. Ce n'est alors plus le langage qui devient complexe, mais ce nouveau paradigme. Penser immuable, fonction et récursif demande du temps. Une fois cette barrière franchie, s'offre alors au développeur le meilleur des deux mondes. Il permet de programmer dans les deux styles, laissant le choix à l'utilisateur de la meilleure solution pour son problème.
La programmation fonctionnelle est plus expressive. Dans un style plus déclaratif qu'une programmation impérative, elle facilite la maintenance, l'évolution et la réutilisation du code.
Pour croître et conquérir le monde, le SI de nos entreprises doit aujourd'hui répondre rapidement à des problématiques complexes. Les systèmes distribués, accessibles en 24/7, ainsi que les calculs parallèles sont devenus la nouvelle norme. L'immuabilité avec la programmation fonctionnelle aide à raisonner dans ces environnements.
La qualité des bibliothèques de l'écosystème Scala en fait aujourd'hui un langage mature et prêt pour le monde de l'entreprise. Des grands acteurs de l'économie du Web comme Twitter, Netflix ou Linkedin ont fait le pari de travailler avec Scala.

Application Windows 8.1 HTML/JavaScript et stockage de données

Cet article présente le développement d'applications tactiles multi-plateformes initialement avec Cordova-PhoneGap et Sencha Touch porté sur Windows 8.1 sans ... Cordova-PhoneGap!

Julie fichet
Ingénieur Développeur chez **VISEO**

C'est à partir d'une application existante Cordova-PhoneGap que nous allons effectivement voir comment réutiliser au maximum le travail réalisé, et déployer cela sur une tablette Windows 8.1, ceci sans passer par la surcouche Sencha Touch. Nous allons également aborder les problématiques de stockage de données.

Possibilités de stockage des données

Découvrez les moyens de stocker des données depuis une application mobile (ici on privilégie les applications Windows 8.x) :

Depuis le navigateur Web

- Le **LocalStorage** et le **Session Storage** ; ces deux types de stockages de données reposent sur un système de clé/valeur qui permet d'enregistrer des données via JavaScript. Le stockage peut aller jusqu'à 5 Mo pour Chrome et Firefox et 10 Mo pour Internet Explorer. Les deux se distinguent par leur type de persistance de données. Le SessionStorage limite l'accès aux données à l'onglet ou à la fenêtre ouverte, ensuite les données sont supprimées. D'autre part, s'agissant du LocalStorage, les données ont une durée de vie illimitée. D'un point de vue sécurité, les données stockées ne sont pas chiffrées, et, par conséquent, toute personne ayant accès à un navigateur peut consulter les données.
- **WebSQL**, n'est pas accessible pour Windows 8 et Windows 8.1 car non pris en compte dans les spécifications de HTML5.
- **IndexedDB** est pris en charge par Internet Explorer depuis la version 10, les applications Windows 8 et Windows Phone 8 y ont donc accès (IE 10 minimum). C'est un stockage d'objets en clé/valeur. Cependant, face à ce mode NoSQL, certaines réticences peuvent apparaître puisque les accès se font sans instructions SQL. Enfin, le volume de données autorisé pour réaliser les sauvegardes dépend également des navigateurs; par exemple, pour Internet Explorer, il n'y a pas de limite fixe ; par défaut cela représente 10 Mo, et sinon 250 Mo pour un « Domain ».

Avec une base « classique » :

- **SQLite :** est une base de données classique, qui n'a pas de limite en termes de taille de stockage maximum. Des plugins existent sur Cordova pour les applications Android et iOS. Cependant, Microsoft n'a pas mis à disposition de tels plugins pour les applications Windows 8.x en HTML/JavaScript.

```
function test(){
var promiseDb = WinJS.Utilities.ready().then(function () {
        return SQLite3JS.openAsync(dbPath);
    });
    promiseDb.then(function (db) {
        var query = "INSERT INTO  test_table (data, data_num) VALUES ('abcdabdbabddbababdbabcdabdbabc
        var promise = insert(db, query);
        var start = new Date();
        for (var i = 0; i < LINES; i++) {
            if (i % 100 == 0) console.log("i: " + i);
            var j = 0;
            promise = promise.then(function () {
                var query = "INSERT INTO  test_table (data, data_num) VALUES ('abcdabdbabddbababdbabc
                j++;
                if (j % 100 == 0) console.log("j: " + j);
                return insert(db, query);
            });
        }

        promise.then(function () {

            db.eachAsync("select count(id) as cnt from test_table;", [], function (row) {
                console.log("row.item(0).cnt: " + row.cnt);
                db.close();
                console.log("database closed");
                var end = new Date();
                console.log("Time:" + (end - start));
            });

        });

    });


}

function insert(db,query)
{
    return db.runAsync(query);

}
```

Testons jusqu'aux limites une solution JavaScript + Windows 8.1 reposant sur SQLite

L'objectif est d'avoir une application pouvant stocker plus de 25 Mo de données. La solution la plus adéquate semble être SQLite. Rappelons que SQLite ne dispose pas de plugin implémenté pour des applications Windows 8 HTML/JavaScript. Aussi, nous allons nous baser sur le travail de Marcus IIgner, qui a développé un plugin en C++ pour utiliser SQLite avec une application Windows 8 JavaScript (lien :https://github.com/doo/SQLite3-WinRT/blob/master/LICENSE).

Pour peupler jusqu'à 25Mo de données, 100000 lignes vont être insérées en base.

A défaut d'utiliser les transactions qui ne sont pas disponibles dans ce plugin, un autre mécanisme sera utilisé : les « promises ». Les objets « promises » permettent de programmer des tâches de manière asynchrone (voir code).

De nombreux essais consécutifs ont été réalisés afin de valider que ces insertions sur la base SQLite s'effectuent convenablement.

Toutefois, comme le révèle le code précédent, il reste à optimiser les insertions. Une boucle de 10000 insertions a été réalisée, et le traitement peut mettre jusqu'à 10 minutes à se terminer.

La meilleure solution serait de faire une seule requête avec toutes les insertions dedans, et d'encapsuler le tout dans une transaction avec db.runAsync('BEGIN TRANSACTION') et db.runAsync('COMMIT TRANSACTION').

Sans framework, tout cela reste fastidieux et très archaïque pour une application qui nécessite des accès à la base.

Pour aller plus loin...

Pourquoi ne pas opter pour la réalisation d'une Application Windows 8 C#/Xaml ? Puisque, dans ce cas le plugin SQLite est implémenté par Microsoft. Dans le cas présent, la problématique est de travailler avec une application déjà existante en Cordova-PhoneGap, où il « suffirait » de reprendre l'existant en termes de code développé. Or, ce framework ne supporte pas les applications Windows 8. Pour éviter une réécriture complète de l'application, le code déjà écrit en JavaScript et HTML (pour Cordova-PhoneGap) a été repris en l'état et inséré dans une application Windows 8 JS. L'application est à présent entièrement opérationnelle dans ce mode.

Arduino, simple comme un circuit imprimé !

A quoi sert l'informatique ? Chacun ira de sa réponse. Pour ma part, m'assister dans mon travail, faciliter ma vie, et, dans le cadre de cet article, me permettre de concrétiser un projet que j'avais en tête depuis quelques semaines mais sans arriver à le mettre en place.

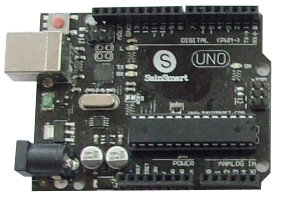
Par Dr César Séjourné

Tout a commencé par une idée très simple : connaître simultanément la température dans toutes les pièces de la maison. Des solutions domotiques existent déjà. En effet, des constructeurs proposent des stations météo permettant de piloter une dizaine de sondes simultanément. Le problème est que la transmission des données repose sur les ondes radios et je souhaitais un système filaire. De plus la fiabilité des données n'est pas toujours au rendez-vous. Il existe des solutions professionnelles, à base de sondes filaires reliées à un serveur TCP/IP, destinées à la gestion de la température des salles serveurs, datacenter, etc. mais leurs tarifs sont prohibitifs. Après des semaines de recherches infructueuses, j'ai dû me rendre à l'évidence ; il fallait que je me débrouille par moi-même. J'ai donc contacté François Tonic, le rédacteur en chef de Programmez, afin de bénéficier de ses conseils. Je lui ai exposé mon projet - connaître la température et l'hygrométrie dans chacune des pièces, via des sondes filaires, avec centralisation possible des données et pourquoi pas une mise à disposition des résultats sur mon réseau intranet – le tout en prenant en compte que je ne souhaitais pas passer mes journées à coder.

Et si on tentait l'aventure Arduino ?

Je pensais que François me proposerait une solution basée sur le Raspberry. J'avais en mémoire l'incroyable projet présenté dans programmez (Programmez n°170 – Janvier 2014) sur son utilisation pour surveiller un jeune chiot. Il y avait aussi l'utilisation des kits Intel (Programmez n°179 – novembre 2014) qui peuvent être reliés à un capteur de température. Que nenni ! Quelle ne fut pas ma surprise lorsque je me suis vu proposer de monter mon projet à l'aide d'un Arduino. A contrario du raspberry Pi, l'Arduino n'est pas un « vrai » ordinateur. Même si la version de base est très ressemblante, avec sa taille inférieure à celle d'une carte de crédit. Là s'arrête la comparaison.

Un kit de démarrage qui vous permettra de vous familiariser avec l'Arduino.



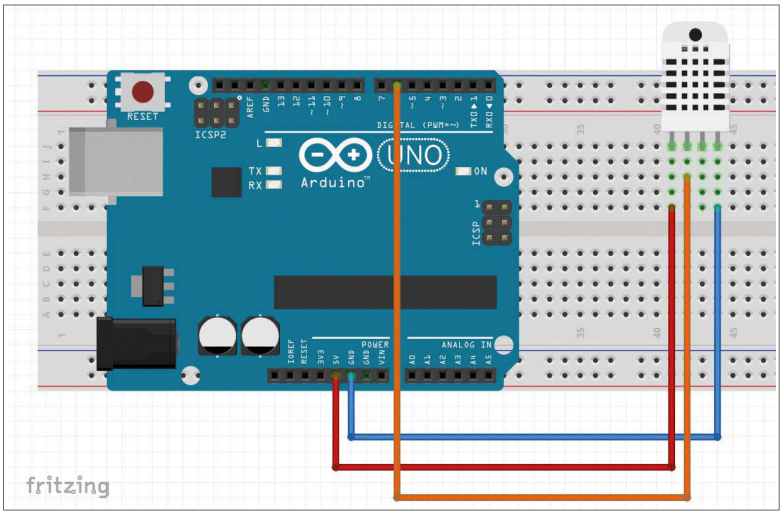
Une copie légale de l'Arduino UNO et son cable USB.

Point de composants habituellement présents dans un ordinateur comme de la ram, un port éthernet, une carte graphique, etc. Non l'Arduino de base est « juste » un circuit imprimé, doté d'un microcontrôleur et de broches d'entrées et de sorties. Ce microcontrôleur est programmable et peut dès lors recevoir des informations et agir via ses broches sur le monde, pour, par exemple allumer une diode ou recueillir la température d'une pièce. L'autre avantage majeur dans mon cas, c'est qu'il n'est pas nécessaire d'installer un système d'exploitation ou de mettre à jour des pilotes, firmwares, etc. Une fois connecté, nous pouvons rentrer directement dans le vif du sujet. D'ailleurs initialement, cet outil était destiné avant tout à des artistes ou des designers afin de leur permettre de mener leurs projets sans requérir à de solides connaissances en électronique.
Fort de ses conseils, je me suis procuré un Arduino, un UNO rev3, coûtant environ 20 € frais de port non inclus sur le site officiel http://store.arduino.cc/ .
Néanmoins les plans sont en licence libre et des copies existent pour bien moins cher. Cette carte est de conception italienne. J'y ai joins un kit, contenant de nombreux capteurs, pour débuter et notamment des capteurs de température. Ainsi armé, j'allais pouvoir déterminer la faisabilité de mon projet.
Il faut tout d'abord relier son Arduino à un ordinateur - MAC ou PC - pour être programmé. Le modèle de base comprend un port USB et le câble est le même que celui utilisé pour connecter une imprimante (certains sont vendus sans câble, il n'y a rien de plus frustrant que de recevoir son nouveau jouet et d'être incapable de s'en servir).
Toutefois, il faut d'abord télécharger le logiciel sur le site officiel en choisissant son système d'exploitation http://arduino.cc/en/Main/Software .
L'installation sous Windows 7 ne pose aucune difficulté. .

Les composants

L'Arduino se code via un éditeur très simple mais efficace : Arduino IDE. Il est très simple d'utilisation. La programmation s'effectue sur une base de langage C++. Les programmes sont appelés sketchs ou croquis et le transfert via l'Arduino s'appelle téléverser. Même sans être un développeur, l'approche est simple.

Pour les capteurs, mon choix s'est porté sur des DHT22. Ils mesurent la température et l'hygrométrie simultanément. Ce choix a été déterminé par un bon rapport qualité prix (merci Amazon), leur simplicité et leur fiabilité (un point important pour moi).
Pour le câblage, les connections sont très simples, puisqu'il n'y a que 3 fils à relier sur l'Arduino : un pour l'alimentation (5 volts), un second pour la masse (GND) et le dernier sur une broche numérique. On peut trouver des schémas sur le web utilisant une résistance. J'ai testé avec et sans. Je préfère sans pour une question de fiabilité des résultats.



Un modèle de câblage avec une sonde.

Un peu de code maintenant

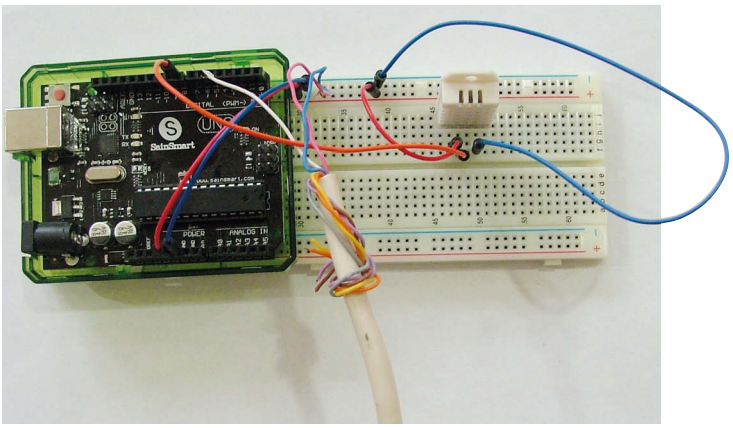
Au niveau programmation, pour pouvoir utiliser les sondes efficacement, il faut récupérer leur librairie et l'inclure dans les sketchs :

```
// déclaration des librairies
#include <dht.h> // librairie pour le capteur de température
```

La déclaration de plusieurs sondes est très simple. Il suffit de déclarer sur quels connecteurs elles sont reliées puis de récupérer les données :

```
// déclaration des sondes
#define DHT22_1_PIN 6 // sonde 1 en pin 6
#define DHT22_2_PIN 8 // sonde 2 en pin 8

// récupération des valeurs de la sonde 1 et affichage
int chk1 = DHT.read22(DHT22_1_PIN);
Serial.print(DHT.temperature,1);
Serial.print(DHT.humidity,1);

// récupération des valeurs de la sonde 2 et affichage
int chk2 = DHT.read22(DHT22_2_PIN);
Serial.print(DHT.temperature,1);
Serial.print(DHT.humidity,1);
```

Globalement c'est simple et à la portée de tous. Par contre, pour mesurer la température dans chaque pièce, il fallait déporter le capteur. J'ai donc utilisé un simple câble téléphonique. Je souhaitais prendre un câble PTT souple de 2 paires torsadées, puisque je n'avais besoin que de 3 fils mais son prix, 2 fois plus élevé qu'un 4 paires torsadées, m'a fait préférer ce dernier. J'ai donc relié une sonde à l'extrémité d'un câble d'une dizaine de mètres. Le résultat fut concluant et la justesse des valeurs fut confirmée par une sonde traditionnelle.



Le montage test : l'Arduino dans un boîtier de protection, une sonde DHT sur un BreadBord directement reliée à la carte et le câble de télécommunication permettant de « déporter » la seconde sonde.

Ce premier test a montré que le choix d'un Arduino dans le cadre de mon projet était pertinent. Sa simplicité d'utilisation lui permet d'être à la porter de tous et le temps de prise en main est rapide. C'est même grisant car de nouveaux horizons s'ouvrent à vous. Vous ne regarderez jamais plus comme avant une guirlande de leds, un détecteur de mouvements, une alarme ou une télécommande.

Tout devient dès lors possible : des capteurs de mouvement et un buzzer et vous obtenez une alarme, un capteur d'hygrométrie terrestre et c'est un système pour contrôler l'arrosage de votre plante préférée. Toutes ces choses qui vous semblaient jusqu'à présent inaccessibles (ou très coûteux) s'ouvrent à vous. Pour revenir à mon projet, je suis en mesure de dispatcher dans toutes les pièces de mon habitation des sondes de température et d'en centraliser les données. Il me reste à transformer mon Arduino en serveur web. C'est tout à fait faisable, puisqu'il est possible de lui connecter des cartes filles (les Shields) pour apporter des fonctions complémentaires (en fait on les empile les unes sur les autres). On trouve des lecteurs de carte à puce SD, des cartes Ethernet, etc. mais cela est une autre histoire.

Bref, allez-y. devenez un maker.

Conseils

Pour débuter et se familiariser, je recommande l'arduino UNO rev3. Il est préférable d'investir dans un kit de démarrage, qui comprend une série de capteurs, de connecteurs, de fils, de résistances et surtout une planche à pain,BreadBoard, pour réaliser les connexions. Vendu sur amazon ou ebay, souvent sans aucune notice, des liens pointant sur des pdf en anglais sont parfois disponibles.
Je conseille aussi l'achat du livre « Arduino pour les Nuls » de John Nussey (24,95 €). Il facilitera la prise en main, offrira un éventail de petits montages à réaliser et permettra de s'approprier au mieux ce nouvel outil. Sans oublier, les forums officiels qui vous seront d'une aide précieuse, voire inestimable : http://forum.arduino.cc/

PROJET R&D RASPBERRY PI

1ère partie

L'informatique a été popularisée par des bricoleurs œuvrant à la confection d'ordinateurs et logiciels dans leur garage ou grenier. Cet aspect débrouillard et créatif s'est peu à peu industrialisé et complexifié. Cependant, beaucoup de développeurs ont gardé leur esprit de Hacker pour transformer les choses en fonction de leurs besoins. Le Raspberry est le prolongement de cet esprit où la seule limite est celle de l'imagination.

Patrick Guillerm
Architecte Java JEE chez **Palo IT**

Le Raspberry Pi est un pico-ordinateur mono-carte. Comme tous les pico-ordinateurs, il a de faibles dimensions (8,56 x 5,4 x 1,7 cm) et poids (45 gr). Sa consommation électrique est également très faible (entre 1,5 W pour le type A et 3,5 W pour le type B). Il n'en reste pas moins un ordinateur complet, avec une sortie HDMI, une prise USB, une sortie audio, etc.

Les différents Raspberry Pi

Il existe deux modèles de Raspberry Pi : le modèle de type A est le plus simple et donc le moins cher (comptez 28 €), alors que le type B propose une meilleure connectique et une mémoire un peu plus élevée (comptez 35 € pour celui-ci). Ce dernier est également doté d'un port Ethernet, ce qui permet de nombreux usages sur le Web comme sur un réseau local. Voici un petit tableau comparatif des deux modèles :

	Type A	Type B
SoC	Broadcom BCM2835	Broadcom BCM2835
CPU	ARM 700MHz	ARM 700MHz
GPU	Broadcom videoCore IV	Broadcom videoCore IV
SDRAM	256 Mo	512 Mo
Ethernet	Non	10/100 Ethernet
Sortie vidéo	HDMI	HDMI
Sortie audio	Stéréo Jack 3,5mm, 5.1 sur l'HDMI	Stéréo Jack 3,5mm, 5.1 sur l'HDMI
Puissance	1,5W (300 mA)	3,5 (700 mA)
Prix	28 €	35 €

Le choix de l'une ou l'autre version dépend principalement de son cadre d'utilisation. Pour tout ce qui est embarqué (radio modélisme, domotique, etc.), le choix du type A est plus avantageux du fait de sa très faible consommation énergétique. Ce type d'utilisation ne nécessite pas une très grande mémoire. Le type B est à privilégier si l'on souhaite avoir une station multimédia, un serveur, etc. Sa connexion Ethernet et sa plus grande capacité en RAM améliorent grandement ce cadre d'utilisation.

Attention à la carte mémoire !

Les cartes mémoires vendues avec les Raspberry PI sont d'assez mauvaise qualité. Il est donc recommandé d'investir dans des cartes mémoires professionnelles comme les SanDisk Extrem Pro ou les Lexar Professional. Elles sont conçues à l'origine pour être utilisées dans des appareils photos professionnels. Elles sont de très grande qualité, aussi bien en raison de leur vitesse d'écriture et de lecture que pour leur résistance. Elles sont conçues pour résister à l'eau, aux rayons-X, aux variations de températures, etc. Bien que leur prix soit assez élevé, cela peut être très désagréable d'avoir une carte mémoire qui cesse de fonctionner le jour d'une démonstration.

Les distributions

Les Raspberry Pi sont dotés d'une architecture ARM : il est donc nécessaire d'avoir un système d'exploitation adapté. Le second point important dans le choix du système est sa consommation de ressources. Comme on a pu le voir, les ressources du Raspberry sont très limitées. Pour ces deux raisons, Linux reste le système le plus répandu sur cette plateforme. Entre les différentes distributions, Debian reste la plus adoptée. Cette distribution, simple à configurer et à utiliser, consomme peu de ressources. La version développée pour le Raspberry Pi s'appelle **Raspbian**. On retrouve également des versions spécifiques à cette architecture pour d'autres distributions comme Fedora (Pidora), Arch Linux, Risc OS. Le choix de la distribution est principalement lié à vos préférences et connaissances de cette dernière.

Certaines distributions sont conçues et optimisées pour un domaine d'application spécifique, c'est le cas par exemple pour RaspBmc, dont le but est de transformer le Raspberry en station multimédia, ou de RaspyFi qui est dédiée à l'audio et au Hi-Fi. Ces distributions viennent avec leurs lots d'applications. Bien souvent, ces derniers sont choisis dans le souci des performances limitées du Raspberry. On retrouve donc tout naturellement les environnements graphiques XFCE ou LXDE. Il en va de même pour les navigateurs Web : on retrouve Midori qui, comme Chrome, est basé sur Webkit mais est bien plus léger. XBMC reste un incontournable sur Raspberry pour transformer cet ordinateur en station multimédia Fig.1.

Concevoir son boîtier

Le monde du Raspberry est celui du DiY (Do It Yourself). Vous pouvez donc donner l'apparence que vous souhaitez à votre boîtier. Pour le concevoir, vous pouvez adopter une méthode couramment utilisée pour réaliser des prototypes qui est de construire le boîtier à l'aide de Lego Fig.2.

Fig.2 *Exemple de boîtier en Lego*

Le site www.framboise314.fr publie régulièrement diverses créations au design plus ou moins élaboré Fig.3.

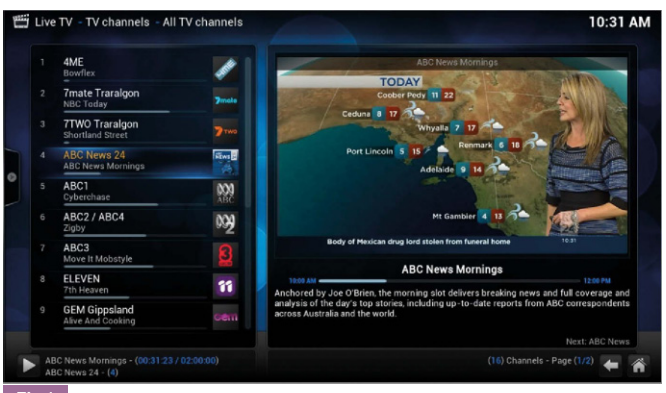


Fig.1 *Interface de XBMC*

Fig.3 *Exemple de boîtier en bois*

Pour certaines formes complexes ou par souci de poids, des matériaux composites comme la fibre de verre ou de carbone offrent de très bons résultats Fig.4. L'avantage de ces matériaux est d'agir comme une toile sur laquelle on applique de la résine époxy. Vous trouverez divers tutoriels sur Internet.

Fig.4 *Exemple de boîtier en fibre de carbone*

PI FACE

Présentation

Une fois notre Raspberry Pi installé dans son boîtier, on se demande ce qu'on peut faire avec. A la vue de la consommation électrique de celui-ci, il semble mieux adapté à de l'électronique embarquée simple. C'est là que Pi Face intervient pour permettre d'interagir avec le monde physique. Il existe plusieurs cartes d'extension :

- **PiFace Digital** : connexion à un interrupteur, une lumière, un moteur, un capteur, etc.
- **PiFace Control and Display** : petit écran LCD et boutons pour contrôler l'état du Raspberry Pi et interagir avec lui sans clavier, ni écran encombrant.
- **PiFace Rack** : permet de connecter plusieurs cartes d'extension. C'est l'équivalent d'un switch réseaux mais pour les cartes d'extension Fig.5.

L'avantage du PiFace est de proposer différentes entrées et sorties du module. On retrouve donc :

- 2 relais : ils permettent de contrôler des systèmes électriques de puissances (comme des moteurs asynchrones).
- 4 interrupteurs : bien qu'accessibles uniquement par le dessus du modules, ils permettent d'apporter des entrées de contrôle au module.
- 8 entrées numériques : elles permettent de connecter des capteurs. Ces capteurs doivent cependant fournir une information binaire (seul deux états sont gérés).
- 8 sorties numériques : comme pour les entrées, seule la gestion de deux états est possible (allumé / éteint). Les sorties permettent de contrôler des systèmes électroniques ou de générer un signal binaire encodé sur 8 bits.
- 8 LED : elles peuvent donner des indications sur l'activité du module. Grâce à un peu de bricolage, on peut également les utiliser comme sorties avec de la fibre optique (en les soudant à l'aide de résine transparente). Cela peut être utile pour isoler totalement le module dans le cas de systèmes à forte puissance (très haute tension ou grand ampérage).

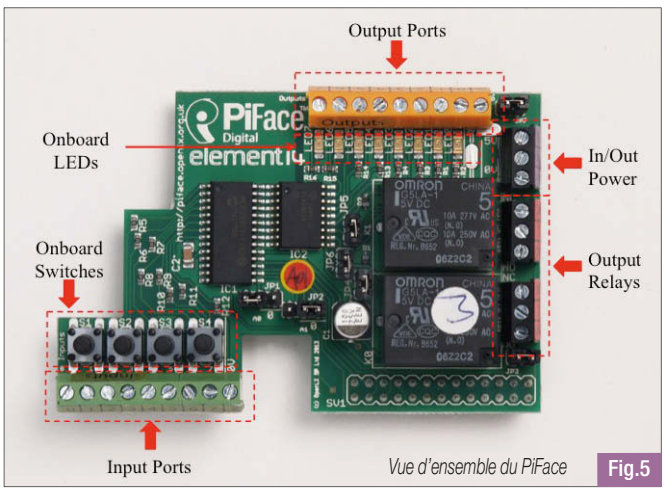


Fig.5 *Vue d'ensemble du PiFace*

Installation

L'installation du PiFace est très simple. Tout d'abord, mettez à jour son système :

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Ensuite, installez les librairies nécessaires à son fonctionnement :

```
$ sudo apt-get install python3-pifacedigitalio
$ sudo apt-get install python3-pifacedigital-emulator
```

Cette commande installera les versions Python 3 pour le PiFace. Bien que les librairies existent pour Python 2, l'émulateur est conçu avec la version 3. Il est donc préférable d'installer la dernière version de Python. Une fois les librairies PiFace installées, il ne vous reste plus qu'à activer le module correspondant dans le Kernel :

```
$ sudo modprobe spi-bcm2708
```

Il est possible d'activer définitivement le module en modifiant le fichier **/etc/modprobe.d/raspi-blacklist.conf** en ajoutant l'instruction **blacklist spi-bcm2708.**

Dès lors, l'utilisateur root a pleinement accès au PiFace. C'est suffisant pour effectuer les premiers tests, mais pour une réelle utilisation, il est préférable d'avoir un groupe d'utilisateurs autorisés à interagir avec le module. Pour ce faire, créez un fichier nommé **50-spi.rules** dans le dossier **/etc/udev/rules.d** avec pour contenu l'instruction suivante :

```
KERNEL=="spidev*", GROUP="spi", MODE="0660"
```

Il ne reste plus qu'à ajouter les utilisateurs du groupe et à leur donner les droits nécessaires :

```
$ sudo groupadd spi
$ sudo gpasswd -a pi spi
```

Pour rappel, sur Raspberry, l'utilisateur a pour login par défaut **pi**. Afin de

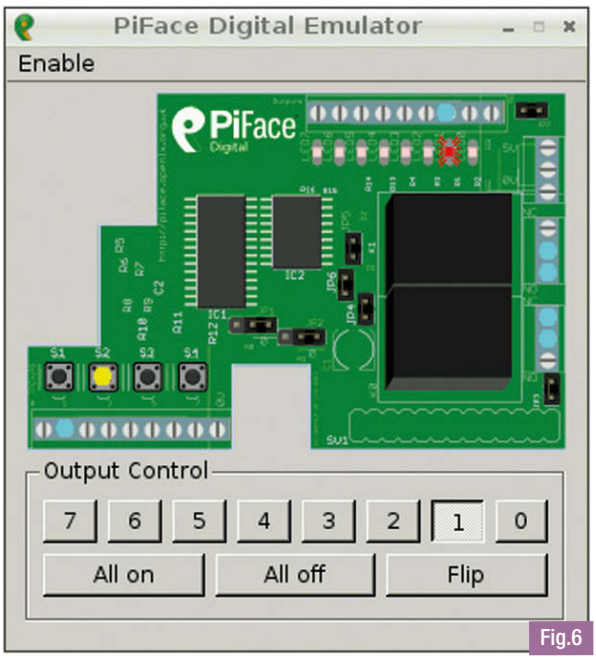


Fig.6 *Screenshot du logiciel PiFace Digital Emulator*

vérifier le bon fonctionnement, il est possible d'exécuter l'émulateur du PiFace avec l'utilisateur **pi** et d'actionner les interrupteurs présents sur le module :

```
$ pifacedigital-emulator
```

Fig.6.

DEVELOPPEMENT

Python

Python est le langage de prédilection du Raspberry PI. Les API Python sont installées en même temps que le module PiFace. Elle comprend également un émulateur de la carte. Il faut cependant faire attention car les API sont développées en Python 3. Nos scripts doivent donc utiliser la même version. Pour pouvoir utiliser l'interface avec le module PiFace, il est nécessaire d'importer la librairie et d'instancier :

```
import pifacedigitalio

if __name__ == "__main__":
    piface = pifacedigitalio.PiFaceDigital()
```

Comme nous l'avons vu, le module PiFace possède différentes entrées et sorties. Elles sont contrôlables via l'API Python :

```
import pifacedigitalio

piface = pifacedigitalio.PiFaceDigital()
# turn on/set high the second LED
piface.leds[1].turn_on()

# turn on/set high the second LED
piface.leds[1].set_high()

# toggle third LED
piface.leds[2].toggle()

# check the logical status of switch3
piface.switches[3].value
>>> 0

# turn on/set high the first relay
piface.relays[0].value = 1

piface.output_pins[6].value = 1

piface.output_pins[6].turn_off()

 piface.input_pins[6].value
>>> 0
```

Dans le cas des entrées, il peut être intéressant de déclencher une fonction à la suite d'un évènement. Cela est possible via les **Listeners.** Cela fonctionne aussi bien sur les entrées que sur les interrupteurs présents sur le module :

```
from time import sleep
import pifacedigitalio

def toggle_led0(event):
  event.chip.leds[0].toggle()

if __name__ == "__main__":
  piface = pifacedigitalio.PiFaceDigital()
  listener = piface.InputEventListener(chip=pifacedigital)
  listener.register(0, piface.IODIR_FALLING_EDGE, toggle_led0)
  listener.activate()

  while True:
    # waiting 1 second
    sleep(1.0)
```

Dans cet exemple, on voit une méthode **toggle_led0** qui va être invoquée à chaque fois que l'entrée 0 est à l'état bas (0), ce qui inversera l'état de la LED 0.
On crée un nouveau **Listener** depuis notre instance PiFace. Une fois créé, il ne reste plus qu'à spécifier notre méthode d'écoute et à activer le **Listener**. On constate que l'API Python est très simple et claire. Elle permet de réaliser des applications interagissant avec le module PiFace sans grande difficulté. Les versions de Python présentes sur Raspberry sont très proches des versions desktop. Dommage que l'émulateur n'en soit pas doté. Il permet de contrôler l'état du module lors des tests et de simuler certaines actions. Pour faciliter le développement, il est préférable de créer une interface qui simulera les différentes entrées et sorties via des appels REST ou clavier par exemple.

Java

> L'API

L'API Java **Pi4J** permet d'interagir avec le PiFace. Cette librairie est disponible sur le dépôt Maven :

- groupId : **com.pi4j**
- artifactId : **pi4j-device**
- version : **0.0.5.**

Bien entendu, le code source est disponible sur GitHub à l'adresse suivante : https://github.com/Pi4J/pi4j/. Cette librairie ressemble beaucoup à celle disponible en Python. On y retrouve les mêmes fonctionnalités avec le coté plus structuré de Java. Il existe cependant quelques subtilités qui diffèrent par rapport à l'API Python. Pour instancier le connecteur PiFace, il est nécessaire d'initialiser deux objets : PiFace et GpioController. L'objet PiFace correspond au module PiFace et GpioController permet de remonter les informations du connecteur entre le PiFace et le Raspberry. Bien que l'objet GpioController ne soit pas forcément utilisé, il est utile qu'il soit initialisé pour le module PiFace :

```
import com.pi4j.device.piface.PiFace;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.wiringpi.Spi;

public class FoobarExample {

  public static void main(String[] args) throws Exception{

    piface = new PiFaceDevice(PiFace.DEFAULT_ADDRESS, Spi.
CHANNEL_0);
    gpio = GpioFactory.getInstance();

  }
}
```

La suite le mois prochain dans Programmez! 182

Les algorithmes de tri

2e partie

En passant à la machine à café, vous avez sans doute déjà croisé des développeurs. Vous avez sans doute constaté qu'ils ont l'air passionnés par leur métier, ce qui rend leurs discussions animées. Et vous les avez sans doute entendu prononcer des mots comme « Bubble », « Quicksort », « logarithme » ou encore « complexité », qui semblent provenir d'une autre langue. Ce sont pourtant des notions primordiales en programmation. Dans cet article, nous allons tenter de démystifier ce charabia.

Thierry Leriche-Dessirier
Architecte JEE freelance / Team leader / Professeur à l'ESIEA
http://www.icauda.com

Pour en finir avec le « tri rapide », je vous propose une version simplifiée de ce qu'on peut lire dans les sources du JDK 6. Dans cette version, il faut regarder avec attention pour bien distinguer la structure de l'algorithme de base.

```
public class RapideCopyJdkTri implements Tri {

  @Override
  public void trier(final int[] tab) {
    if (tab.length == 0) {
      return;
    }
    trier(tab, 0, tab.length);
  }

  private static void trier(final int tab[], final int gauche,
final int taille) {
    final int pivot = tab[gauche];

    int g = gauche;
    int g2 = g;
    int d = gauche + taille - 1;
    int d2 = d;
    while (true) {
      while (g2 <= d && tab[g2] <= pivot) {
        if (tab[g2] == pivot) {
          permuter(g++, g2, tab);
        }
        g2++;
      }
      while (d >= g2 && pivot <= tab[d]) {
        if (tab[d] == pivot) {
          permuter(d, d2--, tab);
        }
        d--;
      }
      if (g2 > d) {
        break;
      }
      permuter(g2++, d--, tab);
    }

    // Swap partition elements back to middle
    int s, n = gauche + taille;
    s = Math.min(g - gauche, g2 - g);
    decaler(tab, gauche, g2 - s, s);
    s = Math.min(d2 - d, n - d2 - 1);
    decaler(tab, g2, n - s, s);

    // Recursively sort non-partition-elements
    if (1 < (s = g2 - g)) {
      trier(tab, gauche, s);
    }
    if (1 < (s = d2 - d)) {
      trier(tab, n - s, s);
    }
  }

  public static void decaler(final int tab[], int a, int b, int n) {
    for (int i = 0; i < n; i++, a++, b++) {
      permuter(a, b, tab);
    }
  }
```

Les gains de chaque version pourraient sembler minimes, en comparaison de l'investissement, mais ils sont réellement importants. Pour bien se rendre compte des différences de performance, voici les temps de traitement moyens observés sur mon ordinateur portable, équipé d'un JDK 7.

Algo	1 000 d'éléments	1 000 000 d'éléments
RapideViaListeTri	15 ms	8 s
RapideViaListe2Tri	13 ms	829 ms
RapideTri	1 ms	909 ms
Rapide2Tri	-	52 ms
RapideCopyJdkTri	-	61 ms
JavaTri (JDK 7)	-	48 ms

On constate que la seconde version à base de liste (RapideViaListe2Tri) est dix fois plus rapide que la première (RapideViaListeTri) pour un million d'éléments à trier, ce qui est loin d'être négligeable. La première version à base de tableau (RapideTri) est presque aussi rapide que la seconde version à base de liste (RapideViaListe2Tri). La seconde version à base de tableaux (Rapide2Tri) est, quant à elle, dix-sept fois plus rapide que la première. Seule la méthode de Java 7 (JavaTriTest), qui est hybride, parvient à faire mieux. Ici, on voit aussi que mon ordinateur est loin de pouvoir effectuer les opérations élémentaires en une nanoseconde puisqu'on se serait alors attendu à une durée de l'ordre de « 20 ms » pour un million d'éléments. Cela est dû en partie à la puissance de la machine et en partie au fait qu'on a du mal à coller au plus près de l'algorithme théorique optimal.

« le choix du bon pivot dépend de la distribution attendue... »

Une des grosses difficultés du « Quick Sort » réside dans le choix du bon pivot, le risque étant de déséquilibrer les sous-listes qu'on va construire. Quand on n'a pas de meilleure idée et faute de mieux, on peut prendre le

premier élément (Fig.5). Lorsqu'on soupçonne que les données sont déjà à peu près triées et que le choix du pivot en première position provoquera un déséquilibre (Fig.6), on peut aussi préférer choisir le pivot au milieu (Fig.7). Et quand on n'a aucune information sur les données, on peut également choisir le pivot de manière aléatoire (Fig.7). Un mathématicien vous dirait que ça permet de maximiser « l'espérance » que ça se passe bien. Personnellement, je n'aime pas trop cette solution. Elle donne souvent de bons résultats, mais le déroulement de l'algorithme est alors aléatoire. Or, en bon informaticien, je n'aime pas trop quand c'est non reproductible. En outre, ça laisse la porte ouverte pour toutes les combinaisons où ça peut mal tourner. Or, en informatique, comme l'énonce la loi de Murphy(6), tout ce qui peut mal tourner va mal tourner Fig.8.

Comme son nom l'indique, le « Quick Sort » a la réputation d'être rapide, ce qui est mérité, mais parfois un peu usurpé. Pour le comprendre, il va encore falloir parler de complexité. Dans le cas favorable où les éléments sont uniformément distribués, c'est-à-dire bien mélangés (Fig.9), l'algorithme produit un arbre équilibré avec « 2 » sous-listes de taille « n/2 » à chaque appel récursif, soit « n » opérations à chaque étape (une étape est un nœud de l'arbre ou une feuille). À l'étape « 1 », on aura donc « 2 » sous-listes. À l'étape « 2 », on aura « 2*2 = 22 » sous listes. À l'étape « 3 », on aura « 2*2*2=23 » sous listes, et ainsi de suite. À l'étape « p », on aura « 2*2*2*...*2=2p » sous listes. L'algorithme récursif s'arrête lorsqu'on arrive à un seul élément. S'il faut « p » étapes pour cela, on pourra donc dire que « n=2p ». La fonction mathématique qui permet de calculer la valeur de « p » quand on connaît « n » est le « logarithme de n en base 2 » qu'on note « log2(n) » ou « lg(n) » en raccourci. Au final, la complexité du « tri rapide » sera donc en « O(n lg n) » Fig.10. Dans le cas défavorable où les éléments sont déjà triés (fig. 10), l'algorithme produit un arbre complètement déséquilibré, ressemblant à une longue tige tordue et ne permettant de trouver la position que d'un seul élément à chaque étape. On aura donc une complexité équivalente à celle du « tri par sélection » en « O(n) ».

Si on raisonne de nouveau en termes de durée, on se rend compte qu'il y a une véritable différence entre un algorithme de tri en « O(n lg n) » et un autre en « O(n) ».

(6) http://fr.wikipedia.org/wiki/Loi_de_Murphy

Nombre d'éléments « n »	Durée pour un tri en « O(n lg n) »	Durée pour un tri en « O(n) »
10	33 ns	100 ns
100	664 ns	10 us
1 000	10 us	1 ms
10 000	132 us	100 ms
100 000	1,6 ms	10 s
1 000 000	20 ms	16 min 40 s
10 000 000	233 ms	27 heures
100 000 000	2,5 s	115 jours
1 000 000 000	29 s	31 ans
Population mondiale	4 min	1 655 ans

Souvenez-vous. Un peu plus tôt, je vous ai dit qu'il fallait « 100 ns » pour trier une liste de « 10 » éléments à l'aide d'un algorithme en « O(n) » et d'un bon ordinateur. Si on se place dans un cas en « O(n lg n) », il ne faudra

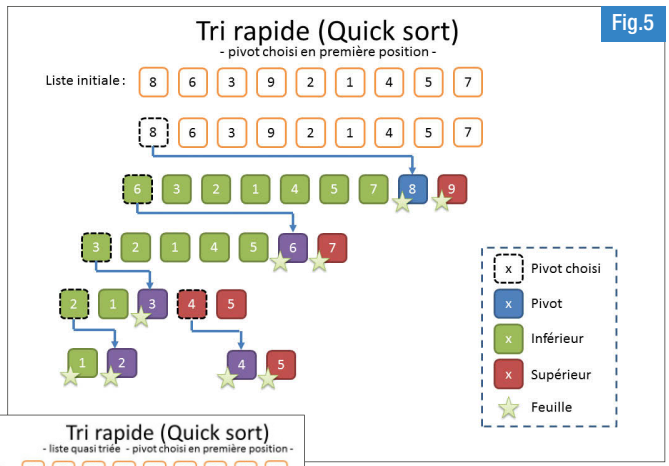


Fig.5

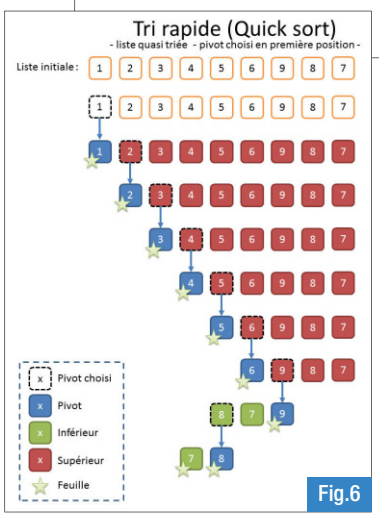


Fig.6

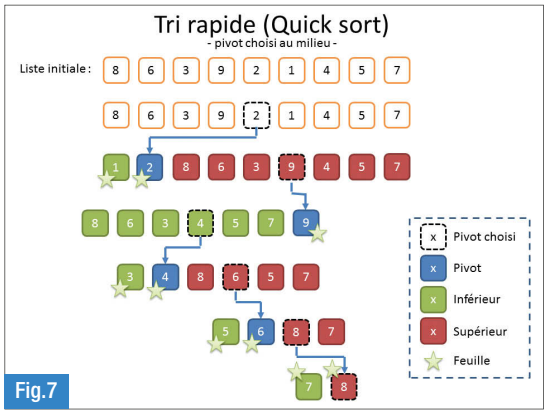


Fig.7

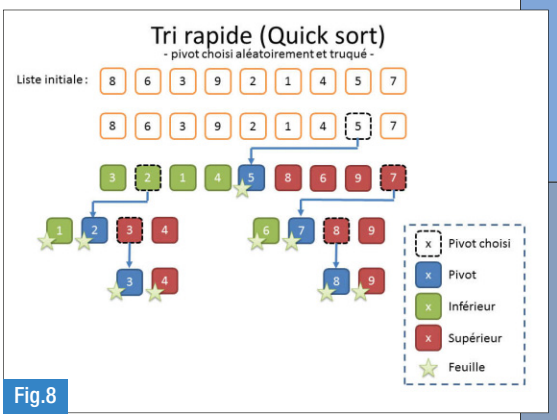


Fig.8

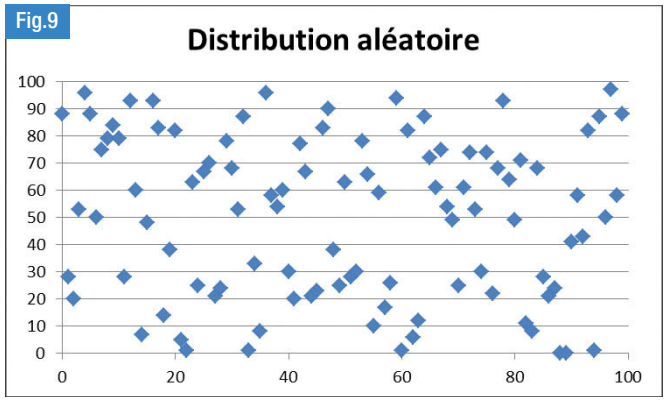


Fig.9

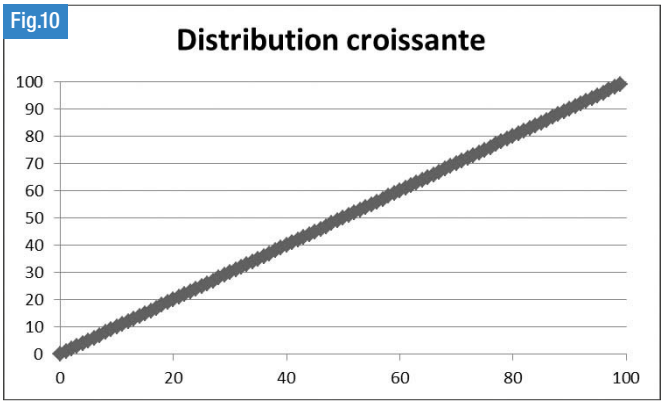


Fig.10

plus que « 33 ns ». Bon, la différence ne saute pas aux yeux. Pour un million d'éléments, on passe de « 16 minutes » à « 20 ms », ce qui reste encore dans les limites du temps réel. Et pour trier la population mondiale, on passe de « 1655 ans » à seulement « 4 minutes ».
Notez qu'il existe une optimisation du « Quick sort » assez contre-intuitive. Elle préconise de mélanger la liste avant de la trier. L'idée est de se rapprocher de la complexité en « O(n lg n) » du cas favorable où la liste est complètement mélangée, quitte à dépenser un « O(n) » à préparer la liste.

Tri fusion (Merge Sort)

Dans la grande famille des tris reposant sur le principe « diviser pour régner », on trouve également le « tri Fusion ». À mon sens, le « tri Fusion » est au « tri rapide » ce que le « tri par insertion » est au « tri par sélection ». Dans cette famille de tri, on travaille toujours en trois phases. D'abord on divise. Ensuite on règne. Et pour finir, on réconcilie (fusionne). Alors que, pour le « Quick Sort », tout se fait à la construction de l'arbre, pour le « tri fusion », la partie intéressante se situe lors de la phase de réconciliation.
Pour réaliser un « tri fusion », on commence par diviser la liste à trier en deux sous-listes de même taille. On réitère récursivement cette opération jusqu'à n'avoir que des listes d'un seul élément. Cela produit donc un arbre à peu près équilibré. On remonte ensuite dans l'arbre en fusionnant les sous-listes à chaque étape. Pour cela, on prend le plus petit élément qui se présente en tête des deux sous-listes à fusionner et on recommence tant qu'il reste des éléments. Quand on revient à la « racine » de l'arbre, la liste est triée Fig.11. Le coût des divisions récursives est quasi gratuit. Il est systématique et ne demande de réaliser aucune comparaison entre les éléments. Si vous avez compris ce qu'on avait dit pour la complexité du « quick Sort », vous avez certainement déjà deviné que la complexité du « tri fusion » sera en « O(n lg n) » dans tous les cas puisqu'on force la production d'un arbre équilibré.

« le tri fusion force la production d'un arbre équilibré… »

Comme pour le « Quick sort », il est plus simple de programmer l'algorithme du « tri Fusion » en commençant par une liste, pour bien le prendre en main.

```
public class FusionViaListeTri implements Tri {

  private List<Integer> trier(final List<Integer> liste) {
    if (liste.size() <= 1) {
      return liste;
    }

    // Séparation en deux sous listes
    final int posCentre = liste.size() / 2;
    List<Integer> listeGauche = liste.subList(0, posCentre);
    List<Integer> listeDroite = liste.subList(posCentre, liste.
size());

    // Tri des deux sous liste
    listeGauche = trier(listeGauche);
    listeDroite = trier(listeDroite);

    // Fusion
    final List<Integer> result = new ArrayList<>(liste.size());

    final Iterator<Integer> iterGauche = listeGauche.iterator();
    final Iterator<Integer> iterDroite = listeDroite.iterator();

    Integer g = next(iterGauche);
    Integer d = next(iterDroite);

    while (g != null || d != null) {
      if (d == null || g != null && g.compareTo(d) < 0) {
        result.add(g);
        g = next(iterGauche);
      } else {
        result.add(d);
        d = next(iterDroite);
      }
    }

    return result;
  }

  private static Integer next(final Iterator<Integer> iter) {
    return (iter.hasNext()) ? iter.next() : null;
  }
```

Pour réaliser le même traitement sur un tableau, il faudra décaler les éléments lors de la phase de fusion des zones triées de gauche et de droite.

```
public class FusionTri implements Tri {

  @Override
  public void trier(int[] tab) {
    trier(tab, 0, tab.length - 1);
  }

  private void trier(int[] tab, int debut, int fin) {
    if (fin <= debut) {
      return;
    }

    // appels récursifs
    final int centre = (debut + fin) / 2;
    trier(tab, debut, centre);
    trier(tab, centre + 1, fin);

    // Fusion
    fusionner(tab, debut, centre, fin);
```

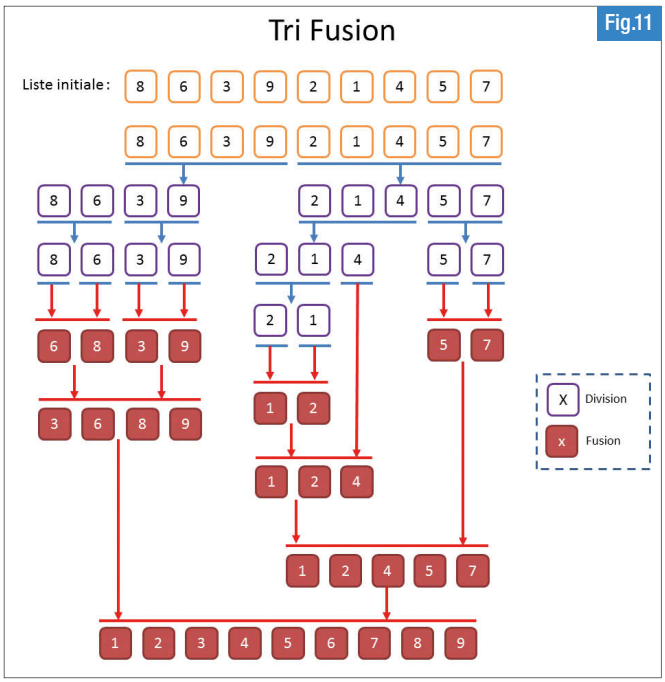


Fig.11

```
}

private void fusionner(int[] tab, int debut, int centre, int fin) {
  int g = centre;
  int d = centre + 1;

  while (debut <= g && d <= fin) {
    if (tab[debut] < tab[d]) {
      debut++;
    } else {
      // Décalage
      int temp = tab[d];
      for (int i = d - 1; debut <= i; i--) {
        tab[i + 1] = tab[i];
      }
      tab[debut] = temp;

      debut++;
      g++;
      d++;
    }
  }
}
```

Tri par interclassement monotone

À l'époque où on utilisait encore des bandes magnétiques, le « tri par interclassement monotone » avait ses adeptes. Comme son nom l'indique, ce tri se base sur la « monotonie » dans l'ordonnancement des éléments d'une liste. Pour dérouler cet algorithme, on répète deux phases jusqu'à ce que ce soit trié, en utilisant des bandes magnétiques ou des zones mémoire temporaires « A » et « B ».

Durant la phase « 1 », on recopie les éléments de la bande magnétique initiale « I » vers la bande « A » tant que les éléments sont croissants. On copie vers la bande « B » dès qu'ils sont décroissants. On continue sur la bande « B » jusqu'à ce qu'il y ait une décroissance en reprenant alors sur la bande « A ». Pour simplifier, on recopie les éléments sur une bande et on change de bande à chaque décroissance.

Durant la phase « 2 », on fusionne les bandes « A » et « B » sur la bande « I » en copiant toujours le plus petit élément en tête des bandes Fig.12.

Je vous vois venir ; on n'en est plus à l'époque des bandes magnétiques. Cela dit, le temps de transfert d'un tableau de la mémoire centrale aux caches CPU, comparé à celui d'une liste chaînée, dont les éléments sont dispersés dans la mémoire, se pose exactement dans les mêmes termes. Et puis, on peut utiliser ce type de tri pour trier des gros fichiers dans un environnement contraint en mémoire puisqu'il n'y a besoin d'ouvrir que deux flux de sortie et un d'entrée, ainsi qu'une paire de variables.

« fonctionnement magique... »

Ce tri fonctionne relativement bien sur des listes mélangées. Son fonctionnement est presque magique quand on s'amuse à le dérouler sur papier. Mais c'est avec les listes en partie triées qu'il dévoile tout son potentiel. La connaissance, même faible, qu'on peut avoir à l'avance sur les données est donc très importante Fig.13. Pour la complexité, très sommairement, il y a donc « n » lectures et « n » écritures par phases. La longueur des sous-suites monotones est TRES grossièrement « 2 » puis « 4 » puis « 8 », bref, on va avoir « lg(n) » étapes (même raisonnement que pour un arbre). C'est assez approximatif, mais disons que dans le pire des cas, c'est en « O(n lg n) » et que, dans le cas de listes partiellement triées, c'est quasi linéaire en « O(n) ». Je vous invite à lire mon blog(7) pour en savoir un peu plus sur le « tri par interclassement monotone » et découvrir quelques-unes de ses variantes.

Positionnement direct (tri sans comparaison)

Parfois, on connait à l'avance la composition de la liste. Imaginons par exemple que la liste soit composée des membres de la Suite de Fibonacci : 1, 1, 2, 3, 5, 8, 13, 21... Avec cette information en poche, on ne va pas perdre notre temps à trier la liste puisque chaque élément porte intrinsèquement sa position. Par exemple, on sait que l'élément « 13 » doit aller à la position « 6 ».

La puissance des machines là-dedans

D'après la loi de Moore(8), et en simplifiant, la puissance des processeurs double tous les 18 mois. Les traitements doivent donc aller deux fois plus vite. C'est vrai, mais c'est sans compter que quantité de données augmente également. Imaginons qu'elle ne fait que doubler sur la même période. Instinctivement, on se dit que ça double la quantité de calcul et que sera compensé par l'augmentation de puissance, mais c'est trompeur. Pour le comprendre, je vais devoir vous infliger encore un petit peu de maths. Prenons un algorithme en « $O(n^2)$ » comme le « tri par sélection ». On double la quantité « n » de données et on divise par deux le temps de calcul pour que ce soit cohérent avec l'augmentation de puissance. On aura donc « $(2n)^2/2 = 4n^2/2 = 2n^2$», ce qui est évidemment plus grand que « n^2». Doubler la puissance ne suffit donc pas à compenser

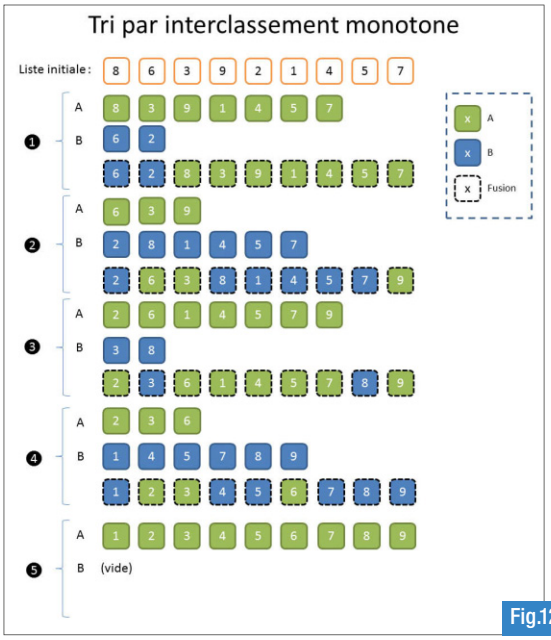


Fig.12

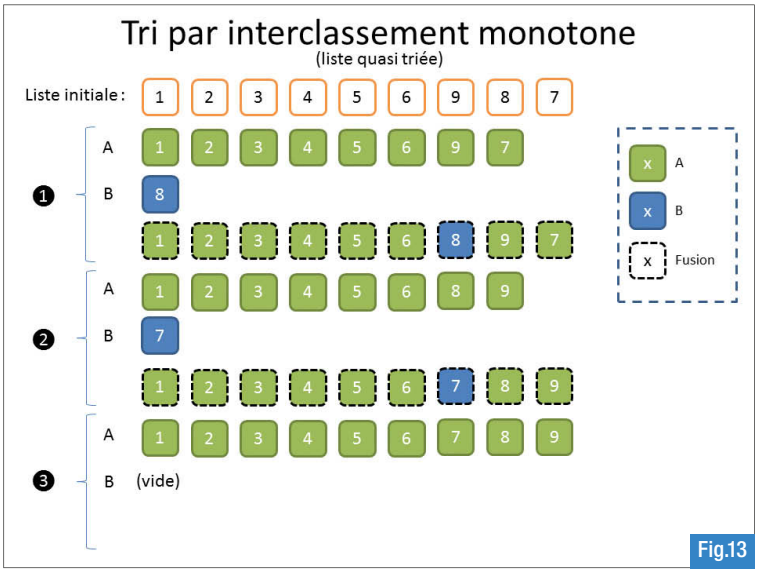


Fig.13

l'augmentation du volume de données et on ne peut donc pas se reposer sur l'amélioration des matériels. Il faut donc choisir avec attention son algorithme et l'adapter à son contexte. D'ailleurs, les constructeurs de processeurs ne font plus la course à la vitesse ; on n'essaie plus d'avoir des MégaHertz, pour plein de bonnes raisons technologiques. À la place, on préfère multiplier le nombre de cœurs. L'avenir des tris complexes passe donc sans aucun doute par le parallélisme offert dans les processeurs multi-cœurs. Des algorithmes comme le « Quick Sort » seront les premiers à en profiter.

Conclusion

Alors, c'est quoi, le meilleur tri ? En fin d'année, Benny Scetbun répondait à une interview(9) à propos de l'école « 42 ». Dans cette interview, il explique que le « Quick Sort » était autrefois considéré (c'est malheureusement encore enseigné à l'école) comme le meilleur algorithme de tri. Il précise toutefois que cela dépend du contexte. En effet, les listes qu'on doit manipuler sont statistiquement déjà ordonnées, tout simplement parce qu'on les a déjà triées la veille, l'avant-veille, etc. Les seuls éléments mal ordonnés sont ceux en queue, qu'on a ajoutés depuis le dernier tri. Du coup, le « Quick sort » est un bon algorithme en théorie, mais un mauvais dans la pratique. L'appliquer sur une liste déjà en partie triée le place dans son pire cas de complexité en « O(n) ».Notez que la bonne stratégie à appliquer dans cet exemple aurait été de ne trier que les données ajoutées récemment puis d'employer un algorithme similaire au « tri par insertion », que je vous ai présenté au début de cet article, pour les positionner correctement.

Combinaisons de tri

Les études des complexités servent d'une part à martyriser les étudiants en école d'info, et d'autre part à savoir quand et comment utiliser les algorithmes. Une des conséquences est qu'il peut être intéressant de combiner plusieurs algorithmes. Par exemple, la méthode de tri incluse dans le langage en Java (mon langage de prédilection) garantit un tri stable avec des performances en « O(n lg n) », ce qui est relativement honnête. Java utilise une combinaison d'algorithmes en fonction de la taille des listes. En dessous de « 7 » éléments, ça utilise un « tri par insertion ». Entre « 7 » et « 40 » éléments, ça emploie un « Quick Sort avec une médiane de trois ». Pour les listes plus grosses, ça utilise un « Quick Sort avec une médiane de neuf ». Les « médianes » désignent des variantes classiques sur le choix du pivot. Notez que les langages évoluent. Ainsi, depuis la version 7, Java utilise un « Quick Sort avec double pivot » pour les listes de plus de « 7 » éléments. Après avoir autant discuté de complexité durant cet article, on pourrait s'étonner de ce choix dans le JDK. En effet, le « tri rapide » est en « O(n lg n) » alors que le « tri par insertion » est en « O(n) ». Or, quand « n » vaut « 6 », le premier vaut « 15 » et le second « 36 », ce qui est bien supérieur. Mais raisonner de cette façon est en réalité une erreur. D'abord, il ne faut pas oublier que, même s'il est vrai que la complexité du « Quick Sort » est en « O(n lg n) » dans le cas favorable, elle tombe en « O(n) » dans le pire des cas (lorsque la liste ou la sous-liste est triée). Ensuite, la complexité doit être considérée uniquement pour des grandes valeurs de « n ». Lorsqu'on manipule des listes petites, il faut dénombrer le nombre réel d'opérations. Pour le « tri par insertion », la formule est « n*(n-1)/2 » dans le pire des cas, comme on l'avait calculé un peu plus tôt. Et il se trouve justement que ça donne « 15 » lorsque « n » vaut « 6 »... Sur le graphe (fig. 15), on observe que les deux courbes se croisent entre les positions d'abscisses « 6 » et « 7 ». Enfin, il faudra noter qu'on retrouve statistiquement de nombreuses sous-séquences déjà ordonnées dans les grandes listes.

(7) http://blog.developpez.com/todaystip/p11899/dev/tri-par-insertion-monotonie
(8) http://fr.wikipedia.org/wiki/Loi_de_Moore
(9) Interview de Benny Scetbun : http://nicotupe.fr/Blog/2013/09/42-interview-de-benny-scetbun/
(10) Podcast Science : http://www.podcastscience.fm

Tri ou presque

La course au meilleur algorithme est un sujet vraiment important pour certaines entreprises. Un bon tri peut même s'apparenter à un avantage concurrentiel sérieux. Dans certaines situations, on n'a pourtant pas réellement besoin d'un résultat parfait. Reprenons l'exemple du photographe avec lequel on a commencé cet article. Il doit trier les enfants selon leurs tailles pour bien composer la photo de classe. Mais en réalité, il ne va pas s'amuser à mesurer chaque élève. Il fait ça au jugé. Et s'il s'est un peu trompé dans l'ordre final, on pourrait parier que ça ne se verra pas.

Tri du dormeur

Pour finir, et parce que je devine que vos yeux se ferment, je voudrais vous présenter un algorithme de circonstance puisqu'il s'agit du « tri du dormeur ». Ce tri a un nom qui me fait rigoler, d'autant qu'il le porte bien, car son principe de fonctionnement consiste précisément à dormir...
Pour chaque élément de la liste, on lance un processus indépendant (un thread), qui se met en sommeil (pause) pour une durée égale à la valeur de l'élément. Par exemple, pour la valeur « 13 », le processus dort durant « 13 » millisecondes. Lorsque le processus se réveille, on ajoute directement son élément (« 13 » dans l'exemple) en queue d'une seconde liste. L'air de rien, ça marche super bien (mille éléments traités en une seconde) pour des listes dont les éléments n'ont pas des valeurs trop élevées. Le « tri du dormeur » nécessite toutefois une quantité astronomique de mémoire et une expérience forte en synchronisation multithread.

Finalement

Bref. On arrive à la fin de cet article, pour de bon. On aurait aussi pu discuter d'autres algorithmes de tri célèbres comme le « tri par base » (« Radix sort »), le « tri par tas » (« Heap sort »), le « tri par dénombrement », le « tri par paquets » (« Bucket sort ») ou encore le « tri shell » dont le principe repose sur des séquences (Pratt, Papernov-Stasevich, Sedgewick) et qui est monstrueusement efficace sur des tableaux presque triés.
En guise de conclusion, je vous invite à les découvrir sur le Web, car ils valent le coup d'œil.

Algorithme	Complexité	Caractéristique
Sélection	O(n)	Interne, stable, sur place
Insertion	+O(n) / -O(n)	Interne, stable, sur place
Bulle	O(n)	Interne, non stable, sur place
Rapide	+O(n lg n) / -O(n)	Interne, non stable, sur place
Fusion	O(n lg n)	Interne/externe, stable, pas sur place
Interclassement monotone	O(n lg n)	
Dormeur	O(a n)	
Base	O(2c (n+k))	Interne, stable, pas sur place
Tas	O(n lg n)	Interne, non stable, sur place
Dénombrement	O(2 (n+k))	Interne, stable, pas sur place
Paquets	O(a n)	Interne, stable, pas sur place
Shell Sedgewick	O(n4/3)	Interne, stable, sur place

En fait, il n'existe pas d'algorithme de tri qu'on puisse considérer comme le meilleur. On pourrait généraliser cela en informatique par le fait qu'il n'y a pas de solution magique qui marche dans tous les cas. Par contre, on connaît des réponses qui fonctionnent relativement bien dans des situations spécifiques.

« il n'y a pas de solution miracle... »

Quand on est développeur, il est indispensable de savoir comment le langage (Java, Lisp...), la base de données (Oracle, MySql, Mongo...) ou encore le progiciel (Kyriba, Excel...) trie les données. Il faut surtout savoir quand utiliser les mécanismes fournis et quand les fuir.
Merci à Étienne Neveu, Fabien Marsaud, Nicolas Tupegabet (Podcast Science(10)) et Olivier Durin pour avoir participé à cet article.

Pattern Acteur : décomplexer la complexité de la concurrence et de la tolérance aux fautes 1ère partie

Dans l'écriture d'une application, la programmation concurrente et la tolérance aux fautes sont souvent des problématiques complexes à mettre en œuvre pour un développeur; en particulier la gestion des threads est souvent fastidieuse pour écrire du code sûr, propre et performant (gestions des verrous, gestions des races conditions, mélange de code technique et de code métier). Nous allons voir que le framework Akka qui implémente le pattern Acteur, répond à cette problématique en fournissant une API de haut niveau, pour faire de la programmation concurrente et de la tolérance aux pannes.


David Hassoun

Le pattern Acteur a été défini en 1973 par Carl Hewitt. Il a été implémenté pour la première fois, dans les années 80, avec le langage Erlang, créé par la société Erickson, pour construire des systèmes télécoms, avec une haute disponibilité. Les Acteurs sont des entités légères, pouvant s'exécuter en parallèle. Un acteur peut être comparé à un objet, à la différence près, qu'il est isolé; il ne partage donc pas son état avec d'autres acteurs. La communication avec d'autres acteurs se fera uniquement par messages et de façon asynchrone. Chaque acteur possède sa propre boîte de réception de messages, à partir de laquelle il effectuera un traitement spécifique en fonction des messages qu'il reçoit.
Ce pattern fournit donc une abstraction de haut niveau, pour faire de la programmation concurrente. Ainsi le développeur n'aura pas à se soucier de problématiques récurrentes de programmation concurrente : manipulation de threads, synchronisation par verrous ... .
Il existe différents frameworks qui implémentent le pattern Acteur dans la plupart des langages de programmation actuels. Nous allons ici nous intéresser au framework Akka, disponible pour le langage Java et Scala.

LE FRAMEWORK AKKA

Akka est un framework open source, créé par Jonas Boner, en 2009. Il est disponible en Java et Scala. Il implémente le pattern Acteur, en se focalisant sur les principes suivants :

- Les acteurs communiquent entre eux uniquement en échangeant des messages de façon asynchrone,
- Les données échangées sont immuables,
- Les acteurs sont localisés de façon transparente (2 acteurs qui interagissent peuvent donc être soit dans la même JVM, soit dans 2 JVM distinctes),
- La notion de supervision pour gérer le cycle de vie des acteurs en cas d'erreur : un acteur est créé et supervisé par son parent. Celui-ci décidera du comportement de son fils lors d'une erreur (arrêt, relance, ...). Cette supervision permet ainsi la création de système qui s'auto réparent.

Architecture

Le framework Akka est organisé en différents modules. Un module principal (akka_actor) contient les fonctionnalités principales pour la mise en place d'un système d'acteurs (création d'acteurs, envoi et réception de messages, supervision, ...). Les autres modules offrent les fonctionnalités suivantes :

- Tests unitaires,
- Distribution d'acteurs sur le réseau,
- Intégration avec des frameworks tiers (Apache camel, ZeroMQ).

Dans cet article nous nous focaliserons sur le module principal.

Création d'un acteur

Pour créer un acteur, le plus simple est d'écrire une classe héritant de la classe **UntypedActor** . La seule méthode à implémenter est la méthode *onReceive*, qui reçoit un message de façon asynchrone et qui effectue le traitement adéquat en fonction du message reçu.

```
public class MonActeur extends UntypedActor {
    public void onReceive(Object message) throws Exception {
        if (message instanceof String) {
            System.out.println("Message recu" + msg);
        }
      }
}
```

Il faut ensuite instancier l'acteur de la façon suivante :
On crée dans un premier temps, un système d'acteurs, de la façon suivante :

```
final ActorSystem systeme = ActorSystem.create("MonApplication");
```

L'objet ActorSystem retourné va alors permettre l'instanciation d'acteur de haut niveau, de la façon suivante :

```
final ActorRef monActeur = systeme.actorOf(Props.create
(MyActor.class));
```

Dans l'instanciation, Akka utilise les **Props** qui permettent d'instancier des acteurs de différentes façons (par factory, par constructeurs, etc ...).
Il est important de noter qu'on ne récupère pas une référence directe à l'acteur instancié, mais un objet **ActorRef** qui référence notre acteur. En effet comme nous l'avons vu précédemment, un acteur peut être localisé de façon transparente sur une autre machine.
C'est pour cette raison qu'on ne récupère pas de référence directe sur un acteur.

Envoyer des messages

Pour envoyer des messages à un acteur, on peut procéder de 2 façons :
En utilisant la méthode d'instance `tell(Object message, ActorRef sender)` qui envoie le message message à l'instance de l'acteur à laquelle elle s'applique.
En utilisant la méthode statique `Future<Object> ask(ActorRef actor, Object message, long millisecondTimeout)` qui envoie le message message à l'acteur actor.
La réponse à ce message, retournée par l'acteur actor sera stockée dans une future.
Cette méthode se basant sur les futures d'Akka, est à utiliser lorsqu'on attend explicitement un résultat après l'envoi d'un message.

Politique de distribution des messages

Pour la délivrance de message, Akka utilise la politique **fire and forget**, c'est à un dire qu'un message sera délivré au plus une fois. Cela signifie que par défaut, il est possible que des messages soient perdus.
Si l'on souhaite assurer une fiabilité dans la délivrance de message (pas de pertes), ce sera au développeur de le gérer (en utilisant un mécanisme d'accusé de réception).

Notion de routage

Akka permet de définir des acteurs de type Router. Ces acteurs reçoivent des messages et les routent vers d'autres acteurs que l'on appelle *routees*. Akka fournit différentes stratégies de routage. Je ne m'étendrai pas sur le routage, mais nous verrons à travers notre étude de cas que cela peut être très utile par exemple pour créer un load balancer d'acteurs.

Etude de cas

Nous voulons écrire un programme qui prend en entrée une liste d'entiers et qui retourne une liste contenant les nombres de la liste d'entrée, multipliés par 2. Cette liste pouvant être grande nous allons utiliser la programmation concurrente via les acteurs pour gagner en performance. Pour cela nous allons utiliser les acteurs suivants :

- Un acteur nommé « Master » dont le rôle sera :
  - De créer un pool d'acteurs nommés « Worker », dont je préciserai le rôle un peu plus loin,
  - De parcourir la liste d'entiers (en entrée) et d'envoyer chaque nombre au pool de workers,
  - De récupérer chaque résultat du traitement effectué par les workers et de les stocker dans la liste à retourner.
- Un acteur nommé Worker dont le rôle sera de calculer le double du nombre qu'il reçoit du master et de lui renvoyer.

Ecriture de la classe Master

Cette classe s'écrit de la façon suivante :

```
public class Master extends UntypedActor {

  /**
   * La liste de nombres en entrée
   */
  private final List<Integer> inputNumbers;

  /**
   * La liste de nombres transformés
   */
  private final List<Integer> transformedNumbers;

  /**
   * Le router
   */
private final Router router ;
  private final long start = System.currentTimeMillis();

  /**
   * Référence vers la future pour lui renvoyer le résultat
   */
  private ActorRef initialSender = null;

  public Master(int nbWorkers, List<Integer> numbers) {
    transformedNumbers = new ArrayList<Integer>();
    this.inputNumbers = numbers;
    router = createRouter(nbWorkers);
    //System.out.println("master and router created");
  }
```

La classe Master est donc instanciée via un constructeur qui entre autres récupère la liste d'entiers à traiter (champ imputNumbers), et crée un routeur avec la méthode createRouter. Le code de cette méthode est le suivant :

```
/**
   * Crée un routeur contenant le pool de workers
   * @param nbWorkers
   * @return
   */
  private Router createRouter (int nbWorkers) {
    Router router = null;
    final List<Routee> routees = new ArrayList<Routee>();
     for (int i = 0; i < nbWorkers; i++) {
       final ActorRef workerRouter = getContext().actorOf
(Props.create(Worker.class));
      getContext().watch(workerRouter);
      routees.add(new ActorRefRoutee(workerRouter));
     }
     router = new Router(new RoundRobinRoutingLogic(), routees);
     return router;
  }
```

Dans cette méthode, l'acteur Master crée un pool d'acteurs Worker. La taille de ce pool a la valeur du paramètre *nbWorkers*. Chaque acteur worker (workerRouter) sera wrappé dans un objet de type **ActorRefRoutee**, qui sera ajouté à la liste *routees*.
Puis le Routeur sera instancié avec la politique **RoundRobinLogic**.
Dans cette classe, il reste à implémenter la méthode onReceive, qui va expliciter le comportement de l'acteur Master en fonction des messages qu'il reçoit.
L'acteur Master, va recevoir 2 types de messages :

- Un message de type **StartMessage**, qui sera envoyé depuis un programme appelant (un programme Main par exemple), pour initier l'acteur Master afin qu'il dispatche les entiers de la liste d'entrée au routeur,
- Un message de type **ResultMessage** qui sera envoyé par un acteur Worker. Ce message wrappe un champ value de type Integer. Ce champ *value* correspond à la valeur calculée par l'acteur worker. Il sera donc ajouté à la liste de sortie (*transformedNumbers*).

Le code de cette méthode est le suivant :

```
@Override
  public void onReceive(Object message) {
    if (message instanceof com.poc.acteur.StartMessage) {
      initialSender = getSender();
      //System.out.println("master received a start event");
      for (Integer number : inputNumbers) {
        // le routeur dispatche les différents éléments de la
 liste au workers
        router.route(number, getSelf());
      }

      //réception du résultat du traitement
    } else if (message instanceof ResultMessage) {
      //System.out.println("master received result event");
      ResultMessage result = (ResultMessage) message;
```

```
      transformedNumbers.add(result.getValue());

      //on détecte quand on a fini le traitement
      if (transformedNumbers.size() == inputNumbers.size()) {

        //calcule la durée du traitement
        Duration duration = Duration.create(System.current
TimeMillis()
            - start, TimeUnit.MILLISECONDS);
        System.out.println(transformedNumbers.size()
            + " numbers computed in " + duration);

        //on renvoie le résultat à la future
        initialSender.tell(transformedNumbers, getSelf());

        // arrete le master et ses workers
        getContext().stop(getSelf());
        getContext().system().shutdown();
      }
    }
  }
```

Pour information, voici respectivement le code des messages échangés :
Le code de la classe StartMessage :

```
package com.poc.acteur;

import java.io.Serializable;

public final class StartMessage implements Serializable{

  /**
   *
   */
  private static final long serialVersionUID = 1L;

}
```

Le code de la classe ResultMessage :

```
package com.poc.acteur;

import java.io.Serializable;


public final class ResultMessage implements Serializable{
  /**
   *
   */
  private static final long serialVersionUID = 1L;

  public ResultMessage(final int value) {
    this.value = value;
  }

  private final  int value;

  public int getValue() {
    return value;
  }
```

Remarque : les informations échangées entre les différents acteurs doivent être immuables. Ainsi il est préférable que les classes correspondant aux messages et leurs attributs soient déclarées avec l'attribut `final`.

Ecriture de la classe Worker
Le code de l'acteur Worker est assez simple à écrire :

```
public class Worker extends UntypedActor {

  private static final int TIMER = 4000;

  @Override
  public void onReceive(Object message) {
    //System.out.println("worker received an event");
    if (message instanceof Integer) {
      int number = (Integer) message;
      int transformedNumber = tranform(number);
      // send result to master
      getSender().tell(new ResultMessage(transformedNumber),
 self());
    }
  }

  private int tranform(int number) {
    try {
      Thread.sleep(TIMER);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    return number * 2;
  }
}
```

Remarque, dans la méthode *transform* du worker, un timer de 2 secondes a été mis volontairement pour montrer que la taille du pool d'acteurs worker; cela permet de diminuer la durée du traitement de la liste d'entiers.

Ecriture du programme principal
Notre programme principal, va effectuer les actions suivantes :
- Initialiser la liste d'entier à transformer,
- Instancier l'acteur Master,
- Envoyer un message de type **StartMessage** à l'Acteur,
- Récupérer la liste transformée via la Future.

Le code source est le suivant :

```
package com.poc.acteur;

import java.util.Arrays;
import java.util.List;

import scala.concurrent.Await;
import scala.concurrent.Future;
import scala.concurrent.duration.Duration;

import akka.actor.ActorRef;
import akka.actor.ActorSystem;
import akka.actor.Props;
import akka.util.Timeout;

/**
```

```
 * Programme principal
 *
 */
public class App
{
 //taille du pool des acteurs de type worker
 public static final int POOL_WORKER = 8;
   public static void main( String[] args ) throws Exception
    {
     //liste d'entiers a traiter
     List<Integer> numbers = Arrays.asList(1,2,3,4,5,6,7,8);
     ActorSystem system = ActorSystem.create("mySystem");

     //création du master
     ActorRef master = system.actorOf(Props.create(Master.
class,POOL_WORKER,numbers));

     Timeout timeout = new Timeout(Duration.create("30
 seconds").toMillis());

   // Création de la future, Acteur implicite auquel notre
master pourra répondre
   Future future = akka.pattern.Patterns
       .ask(master, new StartMessage(), timeout);

   List <Integer> listeResultat = (List<Integer>) Await.result
(future,
       timeout.duration());
     System.out.println("resultat= " + listeResultat);
   System.out.println("nb threads = " + Thread.activeCount());
   system.shutdown();

    }
}
```

Remarque :
La Future utilisée pour récupérer la liste transformée est la Future de l'API Scala et pas celle de l'API standard de Java. Cette Future est en fait un acteur implicite, auquel l'Acteur Master enverra le résultat de retour. C'est pourquoi l'acteur Master, sauvegarde la référence de cet acteur implicite dans la variable `initialSender`.
L'appel à la méthode `ask` est non bloquant. En revanche c'est l'appel à la méthode `Await.result` qui est bloquant.

Exécution

Sachant qu'un timer de 2 secondes a été volontairement mis en place dans la méthode de calcul du worker, nous obtenons les résultats suivants avec une liste de 8 entiers :

- avec un pool de 1 worker nous avons une durée de traitement d'environ 16 secondes (voir trace d'exécution ci-après) :

[INFO] [10/05/2014 19:16:45.158] [mySystem-akka.actor.default-dispatcher-5] [akka://mySystem/user/$a] master and router created with 1 workers
[INFO] [10/05/2014 19:17:01.167] [mySystem-akka.actor.default-dispatcher-4] [akka://mySystem/user/$a] 8 numbers computed in 16009 milliseconds
[INFO] [10/05/2014 19:17:01.168] [main] [ActorSystemImpl(akka://mySystem)] resultat= [2, 4, 6, 8, 10, 12, 14, 16]

- avec un pool de 8 workers nous avons une durée de traitement d'environ 2 secondes (voir trace d'exécution ci-après) :

[INFO] [10/05/2014 19:16:45.158] [mySystem-akka.actor.default-dispatcher-5] [akka://mySystem/user/$a] master and router created with 1 workers
[INFO] [10/05/2014 19:17:01.167] [mySystem-akka.actor.default-dispatcher-4] [akka://mySystem/user/$a] 8 numbers computed in 2013 milliseconds
[INFO] [10/05/2014 19:17:01.168] [main] [ActorSystemImpl(akka://mySystem)] resultat= [2, 4, 6, 8, 10, 12, 14, 16]

La suite dans Programmez ! 182

Ninject, from Zero to Hero

Depuis l'émergence du mouvement Craft, le Clean Code et les principes SOLID sont à la mode. Tout développeur qui se respecte doit produire du code maintenable, testable et réutilisable. Dans cet article, nous allons vous présenter Ninject, un outil dont le but est de faciliter la production d'un tel code. Simple d'utilisation, ce framework modulable se démarque par son interface fluent, de nombreuses fonctionnalités et un grand nombre d'extensions. Après ces quelques pages, vous aurez utilisé Ninject pour faire de l'injection, de l'injection conditionnée, des singletons, de l'AOP, ... Vous aurez toutes les armes pour créer vos applications avec Ninject.

Philippe Lorieul
Consultant chez Cellenza
Blog : http://blog.cellenza.com/

Aurélien Galtier
Consultant chez Cellenza
Blog : http://blogs.developpeur.org/agaltier/

Cellenza

PRÉSENTATION

Dépendance et couplage

Nos programmes sont composés de classes qui « communiquent » entre elles pour exécuter des tâches particulières. Cette communication implique qu'une classe peut avoir besoin d'une autre pour faire son travail. ***Lorsqu'une classe A a besoin d'une classe B pour fonctionner, on dit que B est une dépendance de A. Le degré de dépendance entre deux composants logiciels est nommé couplage.***

Pour illustrer le concept, nous allons prendre l'exemple d'une fonctionnalité utilisée dans la plupart de nos applications : le logging.

```
using System;

namespace GreetingsGenerator
{
  public class GreetingsGenerator
  {
    private ConsoleLogger logger;

    public GreetingsGenerator()
    {
      logger = new ConsoleLogger();
    }

    public void Greet(string name)
    {
      logger.Log(String.Format("Greeting {0}", name));
      Console.WriteLine("Hello {0}", name);
    }
  }
}
```

Dans cet extrait de code, la classe GreetingsGenerator a besoin de la classe Logger pour fonctionner : la classe Logger est une dépendance de la classe GreetingsGenerator. Les dépendances en tant que telles ne posent pas de problème. A vrai dire, on ne peut pas s'en passer. Cela dit, la manière de coder ces dépendances dans nos programmes rend nos logiciels plus ou moins maintenables et évolutifs. Dans l'exemple ci-dessus, le Logger est instancié au sein même de la classe qui l'utilise.

Ceci crée ***un couplage fort*** entre ces classes : on ne peut pas utiliser la classe GreetingsGenerator sans la classe Logger et on ne peut pas changer de logger sans modifier la classe GreetingsGenerator.

Inversion de control et injection de dépendances

L'inversion de control, connu sous l'abréviation IoC, est une technique dont le but est de réduire le couplage entre les composants logiciels. Pour ce faire, les coutures entre les composants logiciels ne sont plus faites dans l'application elle-même, mais par un composant tiers (souvent un framework) jouant le rôle d'orchestrateur.

L'injection de dépendances, ou DI, est une méthode d'IoC. Le principe consiste à « passer » les dépendances aux classes qui les utilisent en évitant les instanciations (les « new ») au sein des classes dépendantes, augmentant ainsi la souplesse du programme. Réécrivons la classe GreetingsGenerator en utilisant l'injection de dépendances

```
using System;

namespace GreetingsGenerator
{
  public class GreetingsGenerator
  {
    private ILogger logger;

    public GreetingsGenerator(ILogger logger)
    {
      this.logger = logger;
    }

    public void Greet(string name)
    {
      logger.Log(String.Format("Greeting {0}", name));
      Console.WriteLine("Hello {0}", name);
    }
  }
}
```

Pour réduire le couplage entre les deux classes, nous avons fait deux choses. Premièrement, nous avons supprimé l'instanciation de la dépendance du service. Elle lui est maintenant passée par constructeur. Ensuite, la classe GreetingsGenerator ne dépend plus directement d'un type spécifique de logger (ConsoleLogger), mais d'une interface. Ceci nous permet d'utiliser n'importe quelle implémentation de logger (Console, fichier, syslog, event windows, etc.). La classe GreetingsGenerator est maintenant beaucoup plus autonome. Elle a besoin d'un service de log mais les détails d'implémentation ne la concernent plus. C'est à présent dans le code client de la classe GreetingsGenerator que se passe la configuration (IoC).

```
using System;

namespace GreetingsGenerator
{
  class Program
  {
    static void Main(string[] args)
    {
      var logger = new ConsoleLogger();
      var generator = new GreetingsGenerator(logger);
      generator.Greet("John Doe");
      Console.Read();
    }
  }
}
```

Notez qu'ici il serait très simple de changer le type de logger passé à la classe dépendante.

```
var logger = new FileLogger() ;
```

Ninject, utilisation simple (cas nominal)

Ninject est un framework d'IoC. Bien qu'il soit possible en pratique de gérer la configuration de vos dépendances « à la main » comme nous venons de le faire, la tâche devient vite laborieuse dans nos applications d'entreprise où les relations de dépendances peuvent être très complexes. Les Conteneurs IoC permettent de centraliser la configuration des dépendances de vos classes. Lorsque vous utilisez Ninject, vous pouvez indiquer à ***un seul endroit de votre application***, que ce soit un fichier de configuration XML ou une classe, quelles implémentations injecter à vos classes. Dans le monde Ninject, l'association entre une implémentation et une interface s'appelle un Binding.
L'installation de Ninject dans vos projet se fait très simplement, à l'aide du gestionnaire de package Nuget.
Une fois le package installé, on peut configurer nos dépendances dans le code. La partie de l'application où est placée cette configuration s'appelle la *racine de composition*. Elle est classiquement située au plus près du point d'entrée du programme (fonction Main pour application console, global.asax pour une application Web, etc.).

```
using System;
using Ninject;

namespace GreetingsGenerator
{
  class Program
  {
    static void Main(string[] args)
    {
      using (var kernel = new StandardKernel())
      {
        kernel.Bind<ILogger>().To<ConsoleLogger>();
        var generator = kernel.Get<GreetingsGenerator>();
        generator.Greet("John Doe");
        Console.Read();
      }
    }
  }
}
```

Regardons ce code de plus près. Il se déroule en 3 temps :

- Récupération d'une référence à Ninject : l'objet kernel représente le conteneur IoC. C'est l'origine du graphe de dépendances. Lorsque l'on veut demander quelque chose à Ninject, on utilise le kernel. Il existe plusieurs types de kernels, mais le kernel standard suffit dans la majorité des cas.
- Configuration des Bindings : une fois que l'on a une référence au noyau de Ninject, on peut configurer les implémentations associées aux interfaces. La ligne kernel.Bind<ILogger>().To<ConsoleLogger>() fait exactement ça : elle dit à Ninject « dès qu'une classe a besoin d'un ILogger, passe lui une instance de ConsoleLogger.
- Récupération de l'instance de service : au lieu d'instancier directement notre service, nous demandons à Ninject de le faire. kernel.Get<GreetingsGenerator().

Comme nous avons préalablement dit à Ninject quel type concret associer à la dépendance ILogger du service, Ninject peut nous fournir l'instance sans problème.

DURÉE DE VIE DES OBJETS INSTANCIÉS AVEC NINJECT

Ninject permet de contrôler de manière très précise la durée de vie – appelée scope – des objets qu'il instancie. On peut lui demander un Singleton, un objet transient, un objet ayant la durée de vie d'un thread, d'une requête ...

Singleton

Au Binding, on a la possibilité de spécifier qu'une classe est un singleton ou pas. Vous n'avez pas besoin de vous en préoccuper au moment où vous programmez si votre classe est singleton ou pas. C'est le moteur de Ninject qui va faire le reste.
Lorsque vous aller faire le binding de votre classe, vous pouvez spécifier un mode singleton :

```
var kernel = new StandardKernel();

kernel.Bind<Logger>().ToSelf().InSingletonScope();
```

Ainsi chaque injection de l'objet Logger se fera avec la même instance. Vous évitez d'utiliser des variables statiques et des locks pour créer votre singleton. Cela vous permet d'utiliser l'injection de dépendance. Vous pourrez lors de vos tests unitaires injecter un objet de substitution.

Scope de Thread

Ninject nous permet de définir des singletons suivant certains contextes. L'un d'entre eux est le thread contexte.

```
var kernel = new StandardKernel();

kernel.Bind<Logger>().ToSelf().InThreadScope();
```

Ce qui aura pour effet de spécifier que chaque injection de l'objet Logger contiendra la même instance tant que vous êtes sur des threads différents. Cela peut permettre d'éviter des problèmes d'accès simultané, que l'on peut avoir avec un singleton. Néanmoins vous créez une instance de classe pour chaque thread.

Scope de Requête

Dans le cas d'une application ASP.Net MVC il peut être intéressant d'initialiser une classe pendant toute la durée du contexte http. C'est assez similaire au contexte d'un thread :

```
private static void RegisterServices(IKernel kernel)
{
    kernel.Bind<Logger>().ToSelf().InRequestScope();
}
```

Pour pouvoir en profiter il faut ajouter le package Nuget Ninject.MVC5. Ce package ajoute la méthode d'extension « InRequestScope ». Ce qui vous assure que l'instance du logger ne sera pas utilisée pour une autre requête http. Et cela vous permet toujours d'utiliser l'injection de dépendance. Vous pouvez par exemple injecter votre contexte de base de données en mode InRequestScope. Cela vous assure d'ouvrir et fermer la connexion à la base de donnée une seule fois pour chaque demande de page Web. Dans le cas où ces différents contextes ne suffisent pas on peut créer son propre scope avec la méthode « InScope ».

```
object obj = new object();
kernel.Bind<Logger>().ToSelf().InScope(ctx => obj);
```

La méthode prend une lambda qui sera interprétée pour déduire l'instance à utiliser au moment de l'injection. Ainsi dans l'exemple on vient de ré-implémenter un singleton car à chaque fois que l'on va évaluer la lambda, la même valeur de object sera renvoyée.

INJECTION CONDITIONNÉE

Au moment du binding on peut spécifier des conditions à Ninject pour déduire la classe à injecter par constructeur. On va par exemple conditionner l'insertion d'une data access factory. Grâce à la méthode « WhenTargetHas<> » on conditionne l'injection.

```
var kernel = new StandardKernel();

kernel.Bind<IDataAccessFactory>()
  .To<DatabaseFactory>()
  . WhenTargetHas<DataBase1>()
  .WithConstructorArgument("database", "DataSource=DataBas
e1;User Id=user;Password=Password");

kernel.Bind<IDataAccessFactory>()
  .To<DatabaseFactory>()
  . WhenTargetHas<DataBase2>()
  .WithConstructorArgument("database", "DataSource=DataBas
e2;User Id=user2;Password=Password2");

kernel.Bind<MyClass>().ToSelf();
```

Dans l'exemple suivant « MyClass » est de la forme :

```
public class MyClass
{
  private IDataAccessFactory dataAccessFactory;

  public MyClass([DataBase1]IDataAccessFactory dataAccessFactory)
  {
    this.dataAccessFactory = dataAccessFactory;
  }
}
```

Ainsi dans l'exemple on utilisera la chaîne de connexion « DataSource=Database1 ;User Id=user ; Password=Password ». Nous avons donc conditionné l'injection en fonction de l'attribut qui a été appliqué sur le paramètre d'entrée. On peut aussi décider que l'injection sera conditionnée par le nom du paramètre notre classe « MyClass » qui ressemblera à :

```
public class MyClass
{
  private IDataAccessFactory dataBase1;

  public MyClass(IDataAccessFactory dataBase1)
  {
    this.dataBase1 = dataBase1;
  }
}
```

Pour que cela marche à la place de la méthode « WhenTargetHas<> » on va utiliser la méthode « When ». Cette méthode permet de spécifier n'importe quelle condition au moment de l'injection.

```
var kernel = new StandardKernel();

kernel.Bind<IDataAccessFactory>()
  .To<DatabaseFactory>()
  .When(ctx => ctx.Target.Name == "database1")
  .WithConstructorArgument("database", "DataSource=DataBas
e1;User Id=user;Password=Password");

kernel.Bind<IDataAccessFactory>()
  .To<DatabaseFactory>()
  .When(ctx => ctx.Target.Name == "database2")
  .WithConstructorArgument("database", "DataSource=DataBas
e2;User Id=user2;Password=Password2");

kernel.Bind<MyClass>().ToSelf();
```

Dans l'exemple, on a conditionné sur le nom du paramètre.
Dans l'exemple précédent on a conditionné l'injection de manière statique. A l'exécution on ne pourra pas changer l'attribut DataBase1 ou changer le nom de la variable. On va maintenant injecter la database factory en fonction d'un paramètre dans l'application.
On va créer une classe DatabaseChooser qui va nous permettre de choisir la database à utiliser :

```
public class DatabaseChooser
{
  public bool IsDataBase1 { get; set; }

  public IDataAccessFactory CreateDataAccessFactory()
  {
    if (this.IsDataBase1)
    {
      return new DatabaseFactory("DataSource=DataBase1;User Id
=user;Password=Password");
    }
    else
    {
      return new DatabaseFactory("DataSource=DataBase2;User Id
=user2;Password=Password2");
    }
  }
}
```

La configuration Ninject ressemblera à :

```
var kernel = new StandardKernel();

kernel.Bind<DatabaseChooser>().ToSelf().InSingletonScope();

kernel.Bind<IDataAccessFactory>()
  .ToMethod(ctx => ctx.Kernel.Get<DatabaseChooser>().Creat
eDataAccessFactory());

kernel.Bind<MyClass>().ToSelf();
```

On créé le DatabaseChooser en singleton pour pouvoir réutiliser la même instance à tout moment. Ensuite on utilise « ToMethod » pour récupérer l'instance de DatabaseChooser et ensuite créer la data access factory.

EXEMPLE D'UTILISATION AVANCÉE :

implémentez votre propre ActionFilterAttribute

ASP.NET MVC offre un mécanisme très utile de décorateur via attribut : les filtres. Ces attributs servent à lancer des traitements avant ou après le déroulement des actions (méthodes publiques des controlleurs) qu'ils décorent. Le système est comparable aux triggers du monde des bases de données.
L'attribute OutputCache par exemple, sert à mettre en cache le résultat d'une action pour une durée donnée.

```
using System;
using System.Web;
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
  public class DataController : Controller
  {
    [OutputCache(Duration=10)]
    public string Index()
    {
      return DateTime.Now.ToString("T");
    }
  }
}
```

Le code obtenu est beaucoup plus simple que celui qu'il aurait fallu écrire sans l'attribut.
Seulement, les filtres sont une spécificité d'ASP.NET MVC et ne sont pas disponibles dans les applications consoles ou WPF. Heureusement, il est très facile d'implémenter un mécanisme similaire à l'aide de Ninject et de deux extensions : Ninject.Extensions.Interception et Ninject.Extensions.Interception.DynamicProxy.
Nous allons voir comment procéder en créant un système de traçabilité des accès aux méthodes sensibles de nos applications.

Traçabilité des accès à une méthode critique

Depuis le Gestionnaire de Package Nuget, installez Ninject.Extensions.Interception.DynamicProxy. Nuget vous propose d'installer les dépendances, dont Ninject.Extensions.Interception
Imaginons maintenant que nous ayons un service offrant des opérations sensibles.

```
public class SensitiveService
{
  public void SensitiveOperation()
  {
    Console.WriteLine("Armement d'un missile");
  }

  public void OtherSensitiveOperation()
  {
    Console.WriteLine("Accès aux informations personnelles
 du Président...");
  }
}
```

Ce que nous aimerions, c'est garder une trace des utilisateurs qui ont déclenché l'exécution de ces opérations. On pourrait bien entendu ajouter une ligne de log dans chaque méthode, mais la duplication de code se ferait sentir rapidement à mesure que le nombre d'opérations sensibles augmenterait.
Nous allons donc créer un décorateur qui se chargera de tracer les accès aux méthodes critiques.

1ere étape : L'interception

Il nous faut tout d'abord créer un intercepteur, c'est-à-dire une classe qui lancera un traitement au moment voulu, dans notre cas, avant l'exécution d'une méthode sensible. Ninject Interception fournit une interface prévue à cet effet : IInterceptor. Cette interface offre une seule méthode : Intercept

```
using Ninject.Extensions.Interception;

namespace GreetingsGenerator
{
  public class SensitiveOperationAccessInterceptor : IInterceptor
  {
    private ILogger _logger;
    private string _userName;

    public SensitiveOperationAccessInterceptor(ILogger logger,
 string userName)
    {
      _logger = logger;
      _userName = userName;
    }

    public void Intercept(IInvocation invocation)
    {
      var methodName = invocation.Request.Method.Name;
      _logger.Log(string.Format("L'utilisateur {0} a effectué \
                    une opération sensible: [{1}]",
                  _userName,
                  methodName));
      invocation.Proceed();
    }
  }
}
```

Notre intercepteur va simplement logger deux informations : le nom de la méthode sensible et le nom de l'utilisateur ayant demandé l'opération. Lorsqu'une méthode sensible s'exécutera, la méthode Intercept de l'in-

tercepteur sera exécutée. Le contexte de l'événement intercepté est représenté par le paramètre invocation et sa propriété Request. C'est cette dernière qui va nous fournir le nom de la méthode critique invoquée. Le logger et le nom de l'utilisateur sont passés par construction et via Ninject, nous allons voir comment un peu plus loin.

2ème étape : Création de l'attribut

Maintenant, il nous faut l'attribut qui va nous servir à décorer nos méthodes. Là encore, l'extension d'interception de Ninject nous apporte le nécessaire : le type InterceptAttribute.

```
using Ninject;
using Ninject.Extensions.Interception;
using Ninject.Extensions.Interception.Attributes;
using Ninject.Extensions.Interception.Request;

namespace GreetingsGenerator
{
  public class TraceSensitiveOperationAccessAttribute :
InterceptAttribute
  {
    public override IInterceptor CreateInterceptor(IProxy
Request request)
    {
      return request.Context.Kernel.Get<SensitiveOperation
AccessInterceptor>();
    }
  }
}
```

InterceptAttribute expose une méthode abstraite CreateInterceptor, qui renvoie une instance de l'intercepteur à utiliser avant l'exécution de la méthode décorée. Le paramètre request représente le contexte dans lequel la méthode est invoquée. Il nous permet d'obtenir une référence au kernel de Ninject. Sans surprise, l'intercepteur qui nous intéresse est celui que nous avons créé précédemment et qui va s'occuper de tracer les accès aux opérations sensibles.

3ème étape : les coutures

Si vous avez fait attention, vous avez remarqué que lors de la création de l'intercepteur, nous ne fournissons pas les paramètres de construction attendus, à savoir le logger et le userName. C'est dans notre racine de composition qu'a lieu la magie....

```
static void Main(string[] args)
{
  using (var kernel = new StandardKernel())
  {
    kernel.Bind<SensitiveOperationAccessInterceptor>()
        .ToSelf()
        .WithConstructorArgument("userName", "John Smith");

    kernel.Bind<ILogger>().To<ConsoleLogger>();

    var sensitiveService = kernel.Get<SensitiveService>();

    sensitiveService.SensitiveOperation();
    sensitiveService.OtherSensitiveOperation();

    Console.Read();
  }
}
```

Voyons le code pas à pas.
D'abord, nous indiquons à Ninject qu'à chaque fois qu'une instance de SensitiveOperationAccessInterceptor lui est demandée, il doit en créer une avec le paramètre userName égal à « John Smith ».

```
    kernel.Bind<SensitiveOperationAccessInterceptor>()
        .ToSelf()
        .WithConstructorArgument("userName", "John Smith");
```

Dans la pratique, le nom de l'utilisateur proviendrait bien évidemment du contexte d'exécution de l'application. Ensuite, nous demandons à Ninject de fournir une instance de ConsoleLogger à chaque fois qu'une instance de ILogger est nécessaire.

```
    kernel.Bind<ILogger>().To<ConsoleLogger>();
```

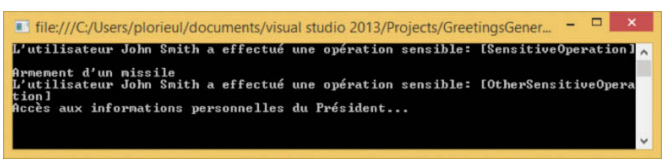
Ces deux instructions nous ont suffi à paramétrer Ninject pour qu'il fournisse l'instance d'intercepteur qui convient dans la méthode CreateInterceptor de notre attribut.
Il ne nous reste plus qu'à récupérer l'instance du notre service à l'aide de Ninject. Attention, si vous instanciez le service vous-même à l'aide du mot-clé new, vous brisez la chaîne d'injection et le système ne fonctionnera pas.

```
    var sensitiveService = kernel.Get<SensitiveService>();
```

Il ne nous reste plus qu'à décorer nos méthodes sensibles... Notez bien que les méthodes décorées de la sorte doivent être virtuelles pour que l'interception fonctionne.

```
public class SensitiveService
{
  [TraceSensitiveOperationAccess]
  public virtual void SensitiveOperation()
  {
    Console.WriteLine("Armement d'un missile");
  }

  [TraceSensitiveOperationAccess]
  public virtual void OtherSensitiveOperation()
  {
    Console.WriteLine("Accès aux informations personnelles
 du Président...");
  }
}
```

Voyons le résultat



Voilà, vous avez implémenté votre propre mécanisme de filtre à la ASP.NET MVC.

Créer des applications mobiles avec Cordova

Le Web a une particularité : c'est une technologie multiplateforme. Le trio HTML5, Javascript et CSS a été pensé pour cela depuis sa création. C'est pour cette raison que dans la quête du Graal du développement d'applications multi plateformes, c'est une option qui revient souvent.

Etienne Margraff
Evangéliste Développeur
Microsoft

Le Web en tant que technologie pour faire des apps est une excellente solution, principalement exploitée de 2 manières :

- Un site Web responsive qui peut aller jusqu'à utiliser des fonctionnalités du téléphone, comme la géolocalisation.
- Incluse dans une application native, dans un composant WebBrowser affiché en plein écran.

Pendant longtemps la première solution était une alternative au fait d'avoir une application disponible sur une plateforme. C'était le « on a pas encore d'app mais d'ici là on a fait une version mobile de notre site ». Ce n'était pas vu comme une solution acceptable, mais plutôt temporaire. Depuis quelques temps c'est réellement considéré comme un choix à part entière comme on peut le voir pour le site TheVerge.com. Les organismes qui travaillent sur les standards fournissent un effort considérable pour rendre disponible la totalité des fonctionnalités du téléphone à une app Web tournant dans le navigateur.
Quand on veut accéder à l'ensemble des fonctionnalités du téléphone il est encore nécessaire à l'heure actuelle d'adopter la seconde solution.
Le fait d'héberger notre code HTML/JS dans un WebBrowser dans une application native nous donne accès à tout, et, accessoirement, rend l'application disponible dans le store des plateformes que l'on cible.
Cette seconde solution est bonne mais quand on vise 3 plateformes (Android, iOS et Windows), on a beaucoup de choses à faire soi-même...
C'est là que **Cordova** entre en jeu !
Apache Cordova est un ensemble d'outils qui :

- Créer automatiquement les applications natives avec leurs WebBrowser pour héberger notre code HTML,
- Fournir un framework Javascript permettant d'accéder aux fonctionnalités du téléphone.

Au travers de cet article, je vous propose de découvrir les bases nécessaires au démarrage avec cette technologie.

Apache Cordova / PhoneGap ?

Au final, quelle est la différence entre PhoneGap et Cordova ?
Il n'y en a quasiment aucune !
A l'origine, PhoneGap a été créé par la société Nitobi. Vue comme une solution excellente pour utiliser HTML et Javascript pour créer des applications, la société est rachetée en 2011 par Adobe. Peu après le rachat, Adobe décide de donner le cœur de PhoneGap en gestion open-source à la fondation Apache. PhoneGap existe toujours et appartient toujours à Adobe, mais constitue une suite de services payants, notamment autour de la compilation multiplateformes.

Les bases de Cordova

Pour démarrer l'utilisation de Cordova, il faut avant tout créer un environnement de développement adapté.
Cet environnement sera constitué de :

- Apache Cordova (http://cordova.apache.org/),
- Les SDKs nécessaires à la compilation pour chaque plateforme :
  - Android : http://bit.ly/androidev
  - Windows : http://bit.ly/windowsdev
  - iOS : http://bit.ly/appleiosdev
- Optionnellement : les émulateurs de chaque plateforme pour déboguer simplement.

> **Note :** *Apple ne permet pas de compiler une application iOS hors d'un environnement Mac, il est nécessaire d'avoir également un Mac pour compiler pour cette plateforme.*

Le plus simple pour télécharger tout ce qu'il faut est d'utiliser NPM (NodeJs Package Manager) que vous obtenez en installant NodeJS (www.nodejs.org). Chaque SDK fournit une documentation complète sur l'ensemble des éléments à récupérer (le SDK Android et ADT pour Windows dans le cas d'Android par exemple).
Une fois que tout est configuré, vous êtes prêts à démarrer votre première application Cordova. Pour cela vous utilisez :

- Le CLI (outil en ligne de commande) de Cordova,
- Un environnement de développement (Notepad++, SublimeText, Visual Studio, xCode, etc.).

L'outil en ligne de commande automatise la création du projet Cordova et la génération des différents projets pour chaque plateforme.

> **Note :** *dans le cas d'iOS et Android, Cordova génère un WebBrowser qui contient le code HTML de l'application. Pour Windows et Windows Phone, c'est un projet natif Javascript qui est créé. C'est exactement le même code HTML que dans le cas des autres plateformes, mais le système le comprend nativement. Il sera par-là légèrement plus rapide à l'exécution sur les plateformes Microsoft.*

Utiliser l'outil en ligne de commandes pour créer un projet

L'outil en ligne de commande est relativement simple à aborder. La création d'un projet démarre par la ligne de commande suivante : **Fig.1.**
Cela a pour effet de créer un répertoire qui correspond à l'application Cordova. Ce projet contient 3 répertoires et un fichier config.xml.
Le répertoire **www** est là pour héberger l'intégralité du code de votre application. C'est ici que vous déposerez les fichiers HTML, Javascript, CSS et tout ce qui est nécessaire au fonctionnement de votre application. Dans une application Cordova, la Webview affiche un contenu Web localisé dans le package de chaque plateforme (Android, iOS, Windows) et non pas téléchargé depuis Internet.
Le répertoire **plugins** contiendra l'ensemble des plugins que vous ajouterez à l'application. Un plugin permet d'activer une fonctionnalité, comme par exemple l'accès aux contacts du téléphone.
Le répertoire **platforms** quant à lui est destiné à héberger le code temporaire qui permet de générer les applications natives que vous ciblez pour votre projet. Vous y retrouverez un projet xCode si vous ciblez iOS, un projet Java pour Android et un projet Visual Studio dans le cas de Windows.

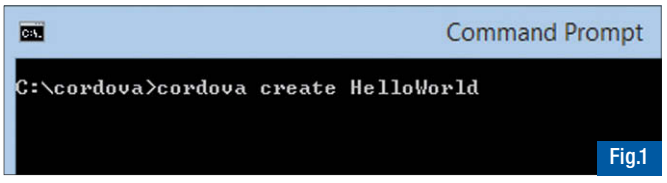


Fig.1

Ajouter une plateforme est justement l'étape suivante. Vous faites cela très simplement avec la commande ***cordova platform add [PLATFORM]*** ou [PLATFORM] est à remplacer par ***ios***, ***android*** ou ***windows***. Voici un exemple pour les projets pour Android et Windows : Fig.2.

> **Note :** *Attention à bien vous positionner dans le répertoire du projet que vous avez créé avec la commande précédente.*

Pour compiler les projets fraîchement créés dans le répertoire **platforms,** si vous le voulez, vous pouvez utiliser les environnements de développement associés. Cependant, Cordova vous simplifie la vie en proposant de faire cela directement à partir de l'outil en ligne de commande en écrivant ***cordova build [PLATFORM]***. Par exemple, pour compiler sur Android vous exécutez : **cordova build android.**
Cette commande qui permet de compiler, exécute en réalité 2 commandes successivement :

- ***Cordova prepare [PLATFORM]*** : copie le contenu du répertoire www dans le projet de la plateforme,
- ***Cordova compile [PLATFORM]*** : compile le projet pour la plateforme.

Lorsque le projet est compilé pour une plateforme, vous avez deux possibilités pour le tester. Vous pouvez le déployer et l'exécuter dans un émulateur, ou le déployer et l'exécuter sur une machine physique (un téléphone, une tablette, ou un PC dans le cas de Windows). Et bien entendu, il y a une ligne de commande pour ça :

- ***Cordova emulate [PLATFORM]*** : déploie dans un émulateur
- ***Cordova run [PLATFORM]*** : exécute sur un périphérique physique (connecté en USB)

> **Note :** *il faut installer les pilotes nécessaires et configurer votre téléphone pour qu'il soit accessible en développeur (autrement dit : que vous puissiez l'utiliser pour déployer une application de développement).*

Ci-dessus, l'exemple de base généré par Cordova qui tourne dans un émulateur Windows Phone. Fig.3.

Accéder aux fonctionnalités du téléphone

Désormais vous savez comment exploiter la moitié de l'intérêt de Cordova : générer automatiquement une application qui contient votre code HTML. La seconde moitié, c'est la possibilité d'accéder aux fonctionnalités du téléphone. Pour cela on utilise les plugins dont vous pouvez trouver la liste complète sur le site de documentation de Cordova : http://bit.ly/cordovaplugins.

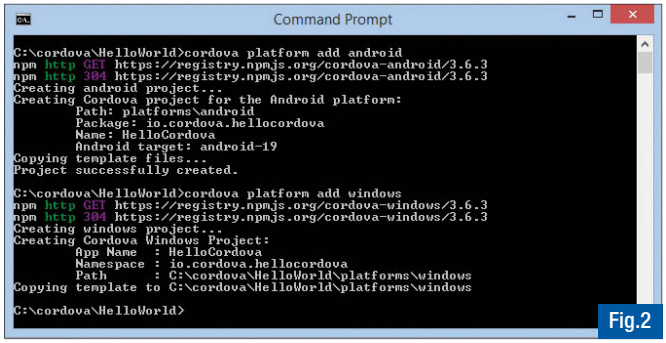


Fig.2

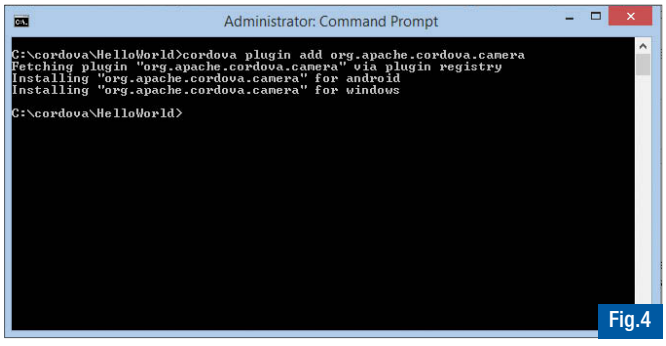


Fig.4

Fig.3

Pour illustrer l'utilisation des plugins, nous allons ajouter un bouton dans l'exemple de base de Cordova qui va lancer le logiciel de l'appareil photo.
Dans un premier temps, on ajoute le plugin en exécutant la commande : ***cordova plugin add org.apache.cordova.camera***.
Chaque plugin possède un nom propre et doit être ajouté individuellement. Notez que le code du plugin est téléchargé et ajouté pour chaque plateforme présente (et pour celles que vous ajouterez par la suite). Fig.4. Pour utiliser cette fonctionnalité on modifie le fichier **index.html** qui est situé dans le répertoire **www**. Le code HTML contient une référence vers deux fichiers Javascript.
index.js contient la logique de la page. Pour l'instant, il est pré-rempli avec la gestion de certains évènements de la vie de l'application (par exemple : **ondeviceready** qui est exécutée quand Cordova est chargé complétement). **cordova.js** n'existe pas sur votre disque. Du moins, pas dans le répertoire **www**. C'est en réalité un fichier qui est généré pour chacune des plateformes lors de la préparation du résultat en ligne de commande. Il contient l'API javascript dédiée à la plateforme pour donner accès aux fonctionnalités du téléphone et gère les plugins et le cycle de vie de l'application.
Dans le fichier index.html, on a supprimé le code situé dans la balise * * et remplacé par :

```
<button id="showCamera">Prendre une photo</button>
<img id="picture" style="height: 100px; width:80px;" />
```

Dans le fichier index.js, la fonction *receivedEvent* a été supprimée et *onDeviceReady* modifié de la manière suivante :

```
onDeviceReady: function () {

  document.getElementById('showCamera').onclick = function () {

    navigator.camera.getPicture(
      function (pictureData) {
        //Afficher l'image dans le tag img

        var picture = document.getElementById('picture');
        picture.src = pictureData;

      },
      function (error) {
        //TODO : gérer l'erreur
      },
      null);

  };

}
```

La fonction ***navigator.camera.getPicture*** n'est accessible que si le plugin de la caméra est ajouté au projet et elle appelle l'appareil photo du téléphone. Elle prend 3 paramètres :

- Une fonction de retour une fois la photo prise,
- Une fonction en cas d'erreur,
- Un ensemble de paramètres.

La fonction de callback récupère le chemin dans le téléphone vers la photo qui vient d'être prise. Dans notre exemple, on l'affecte à l'image de la page HTML. C'est un exemple d'utilisation très simple et certainement à perfectionner mais cela illustre bien le fonctionnement des plugins. Chaque plugin est documenté sur le site de Cordova.

Un environnement de dev adapté

Jusqu'à présent, tout ce que nous avons réalisé est fait en ligne de commande. Est-ce réellement obligatoire ? Doit-on vraiment connaître le nom de chaque plugin ? Comment faire pour déboguer en pas à pas ? Microsoft Open Technologies (http://msopentech.com/) investit depuis un certain temps dans le projet Cordova en tant que contributeur. C'est dans la suite logique de cet investissement que cette équipe fournit l'extension **Multi-Devices Hybrid Apps**. C'est un complément à Visual Studio, l'outil de développement de Microsoft.

Les principaux avantages de cette extension sont :

- Le téléchargement et l'installation automatique de tout ce qu'il faut pour compiler et tester des applications Windows, iOS et Android,
- L'automatisation de l'ajout des plateformes et des plugins,
- La compilation et le déploiement à distance sur un Mac grâce à un outil en ligne de commande créé par l'équipe,
- Le debug pas à pas sur Android, soit dans un émulateur, soit sur un téléphone ou une tablette connectée en USB,
- Et bien évidemment : l'ensemble des fonctionnalités de Visual Studio pour le refactoring, l'intelliSense, etc.

Une fois que l'extension est ajoutée, un nouveau type de projet est disponible : Fig.5.

> **Note :** *il est également possible d'utiliser TypeScript comme langage de programmation pour ce type de projet. Celui-ci vous permet de créer un code Javascript propre et standard, tout en utilisant des concepts de langages typés. Pour en savoir plus :* http://www.typescriptlang.org/ .

Ce nouveau type de projet dans Visual Studio n'est pas une manière ''Microsoft'' de générer un projet Cordova : c'est simplement un outil qui vous facilite la vie tout en respectant les concepts de la technologie. Le projet généré contient le code HTML/Javascript/CSS de l'application et n'est autre que le répertoire **www** avec lequel vous êtes maintenant familier. Le fichier config.xml contient certaines informations du projet ainsi que les plugins que vous voulez utiliser. Visual Studio vous permet de l'éditer graphiquement. Par exemple, pour ajouter un plugin, il suffit de le cocher et l'outil exécutera la bonne ligne de commande pour vous ! Fig.6. Pour lancer l'application sur un émulateur, un téléphone ou une tablette, il faut sélectionner la solution et changer la propriété **Active Config** vers la cible de votre choix. Lancer en débug sur un émulateur Android équivaut à déployer l'application, l'exécuter, attacher le débogueur de Visual Studio et l'explorateur de code dom HTML... tout cela en appuyant sur **F5**. Fig.7. Dans le cas d'iOS, une petite configuration sur un Mac vous permet de lancer la compilation et le déploiement sur un iPad à partir de Visual Studio depuis votre PC.

Grâce à cet outil, vous pouvez facilement piloter votre application Cordova sur toutes les plateformes à partir d'un seul environnement.

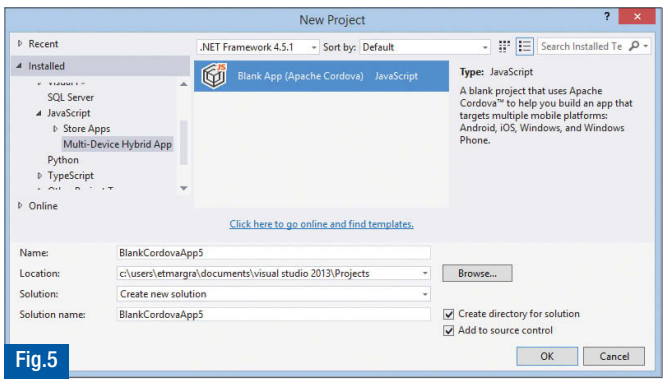


Fig.5

Pour aller plus loin

Maintenant que votre environnement de développement est prêt vous pouvez démarrer votre application, mais quand on créé un projet Web c'est très rare que l'on parte de zéro.

Côté Javascript on utilise au minimum jQuery pour naviguer facilement dans le dom. On ajoute souvent un framework comme Knockout pour le binding, voir AngularJS pour mettre en place une architecture MVC. On peut même considérer des framework plus récents, comme WinJS 3.0 qui permettent d'obtenir des contrôles modernes et évolués.

> **Note :** *si vous voulez utiliser les plugins de Cordova en exploitant les promesses d'AngularJS, vous pouvez utiliser le projet open-source :* http://ngcordova.com/

Et n'oubliez pas qu'une application Web mobile, c'est une application avant tout responsive. Le même HTML est affiché sur plusieurs tailles d'écrans. Pour ça pensez votre ergonomie et la structure de vos pages pour permettre cette gymnastique. Vous pouvez même envisager l'utilisation de Bootstrap pour vous faciliter le travail.

Si vous voulez aller plus dans les détails, je vous invite à suivre le cours que j'ai mis à votre disposition sur la plateforme Microsoft Virtual Academy, qui vous guide à travers la création d'une application Cordova en utilisant AngularJS et Bootstrap : http://bit.ly/apprendreCordova

Au final, Cordova est une très bonne solution pour créer une application multi plateformes en exploitant ses compétences Web. Attention cependant aux performances réelles et perçues : jouez avec les styles CSS notamment pour donner l'illusion d'une application native en retirant certaines animations superflues.

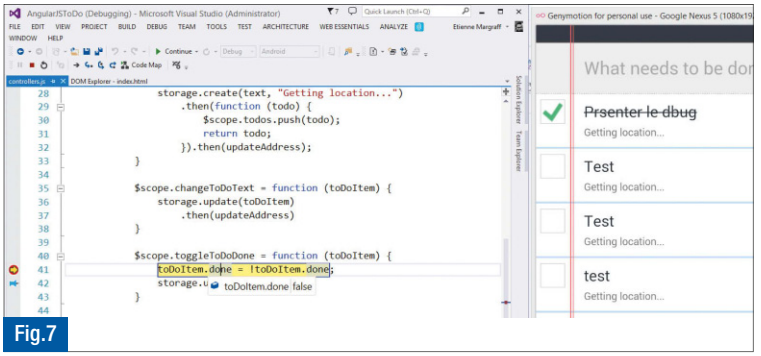


Fig.7

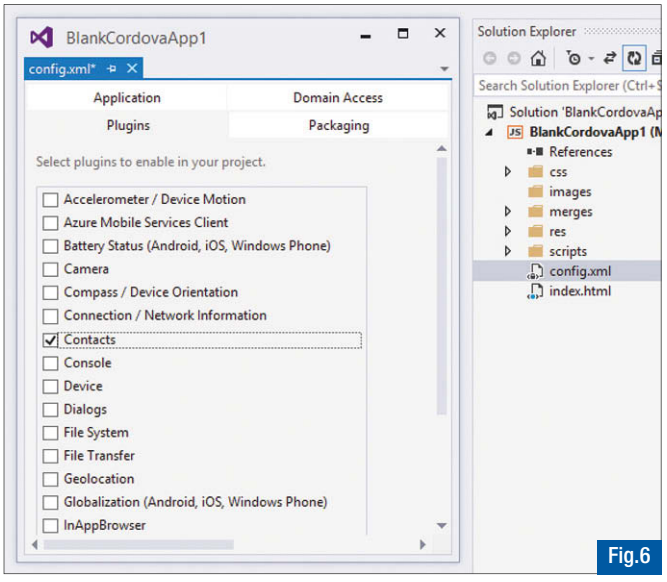


Fig.6

Débuter avec FileOpenPicker sur Windows (Phone 8.1 Silverlight et Windows Phone 8.1)

La classe FileOpenPicker existe depuis Windows 8 mais n'est apparue sur le SDK de Windows Phone que depuis la version 8.1. Auparavant, sur les SDK antérieurs, nous utilisions par exemple PhotoChooserTask pour ouvrir une photo stockée sur notre smartphone. Cette classe fonctionne toujours mais est quelque peu limitée...

Christophe Peugnet
MVP Windows Plateform Devlopment chez
SodeaSoft / EBLM http://www.sodeasoft.com
Blog : http://www.peug.net
Twitter : @tossnet1

En contrepartie de la limitation de **PhotoChooserTask** son utilisation est très simple et se résume à ce petit bout de code :

```
var photoChooserTask = new PhotoChooserTask();
photoChooserTask.Completed += photoChooserTask_Completed;
photoChooserTask.Show();
```

La classe **FileOpenPicker, beaucoup plus riche que PhotoChooserTask,** s'avère tout de même quelque peu plus ardue à mettre en œuvre. D'autant plus que son implémentation diffère si on est dans une application universelle ou en Silverlight.
Le bon côté est qu'elle offre donc plus de possibilités, comme celle de filtrer les types de fichiers affichés par leurs extensions, de laisser l'utilisateur choisir son emplacement... Elle est donc idéale pour les smartphones avec une carte SD ou encore pour prendre directement une photo ou une vidéo et de la récupérer.
Même si **PhotoChooserTask** fonctionne toujours (*et bien heureusement*), je vous conseille tout de même de lâcher tout doucement Silverlight lorsque je vois arriver Windows 10 ; conseil d'ami.
Lorsque l'on utilise **FileOpenPicker** notre application est temporairement suspendue et est réactivée une fois votre sélection terminée ; c'est là que vous constaterez que le code n'est plus aussi facile que **PhotoChooserTask**.

Windows Phone 8.1 "Silverlight"

Mieux que des discours, voici le code : Dans l'app.xaml.cs :

```
public FileOpenPickerContinuationEventArgs FileOpenPicker
ContinuationArgs { get; set; }
private void Application_ContractActivated(object sender,
 IActivatedEventArgs e)
{
    var fileOpenPickerContinuationArgs = e as FileOpenPicker
ContinuationEventArgs;
    if (fileOpenPickerContinuationArgs != null)
    {
this.FileOpenPickerContinuationArgs =      fileOpenPicker
ContinuationArgs;
    }
}
```

Dans une page, placez un bouton et une image pour visualiser le résultat. La commande s'appelle comme ceci :

```
void Button_Click(object sender, RoutedEventArgs e)
{
    var op = new FileOpenPicker();

    op.ContinuationData["Operation"] = "LoadPicture";
    op.FileTypeFilter.Add(".jpg");
    op.FileTypeFilter.Add(".png");
    op.PickSingleFileAndContinue();
}
```

La fonction **ContinuationData** n'est pas indispensable ici mais si vous utilisez plusieurs Picker, cela va permettre d'identifier l'appelant lorsque vous récupérerez la sélection.

Pour récupérer l'information de l'interface Picker ; Ajouter dans le OnNavigationTo de la page comme ceci :

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
  base.OnNavigatedTo(e);

  var app = App.Current as App;
  if (app.FileOpenPickerContinuationArgs != null)
  {
  this.ContinueFileOpenPicker(app.FileOpenPickerContinuationArgs);
  }
}
```

Et dans la méthode *ContinueFileOpenPicker* , vous pourrez faire ce que vous voulez avec le fichier, ici je l'affiche :

```
async void ContinueFileOpenPicker(FileOpenPickerContinuation
EventArgs args)
{
  if ((args.ContinuationData["Operation"] as string) == "Load
Picture"
    && args.Files.Count > 0)
  {

    var imgFile = args.Files[0];

    var stream = (IRandomAccessStream)await imgFile.Open
Async(FileAccessMode.Read);

    var bitmapImage = new BitmapImage();
    bitmapImage.SetSource(stream);
    this.MonImage.Source = bitmapImage;
  }
}
```

En Windows Phone 8.1

Il va falloir ajouter à votre projet la classe **ContinuationManager** qui est disponible dans l'exemple du site MSDN à cette page
http://go.microsoft.com/fwlink/?LinkID=39412
Au niveau de l'App.xaml.cs cela se passe dans la méthode **OnActivated** :

```
protected override void OnActivated(IActivatedEventArgs args)
  {
    base.OnActivated(args);

    var rootFrame = Window.Current.Content as Frame;
    var mainPage = rootFrame.Content as MainPage;
    if (mainPage != null && args is FileOpenPickerContinuation
EventArgs)
    {
      mainPage.ContinueFileOpenPicker(args as FileOpenPicker
ContinuationEventArgs);
    }
  }
```

Sur une page où vous placerez un bouton et une image pour visualiser la sélection, il ne faut rien mettre dans le OnNavigatedTo mais tout simplement implémenter l'interface IFileOpenPickerContinuable comme ceci :

```
public sealed partial class MainPage : Page, IFileOpen
PickerContinuable
```

Le reste (la commande au niveau du bouton et la méthode *ContinueFileOpenPicker* ne change pas.

Les *FileTypeFilter*

Dans l'exemple, vous avez pu remarquer que j'ai ajouté deux filtres d'extensions pour visualiser les images .PNG et les .JPG. Si je n'ajoute que le filtre sur les .PNG, il faut que ce filtre ne supprime pas visuellement les images .JPG mais il m'empêchera de les choisir ! Le résultat sera comme la capture ci-dessous : Les images .JPG sont présentes mais je ne peux les sélectionner : Fig.1.

Capturer une vidéo

Petite astuce, **FileOpenPicker** permet de capturer une vidéo; si certains rêvaient d'une fonction « VideoChooserTask » eh bien en WP8.1 c'est quasi chose faite. Il suffit de mettre ce filtre :
openPicker.FileTypeFilter.Add(".mp4");
En appuyant sur l'icône de l'appareil photo de l'UI Picker en bas de l'écran c'est automatiquement la fonction vidéo de la caméra qui sera activée : Fig.2 et 3.
Voilà, cela sera le minimum vital pour cette fonction et j'espère que cela vous mettra en appétit. Ensuite, **File Picker** permet aussi de faire des sélections multiples, de sélectionner un dossier et d'enregistrer un document.
Je vous invite à découvrir en détail cette classe sur MSDN :
http://msdn.microsoft.com/fr-fr/library/windows/apps/windows.storage.pickers.fileopenpicker.aspx

Fig.1

Fig.2

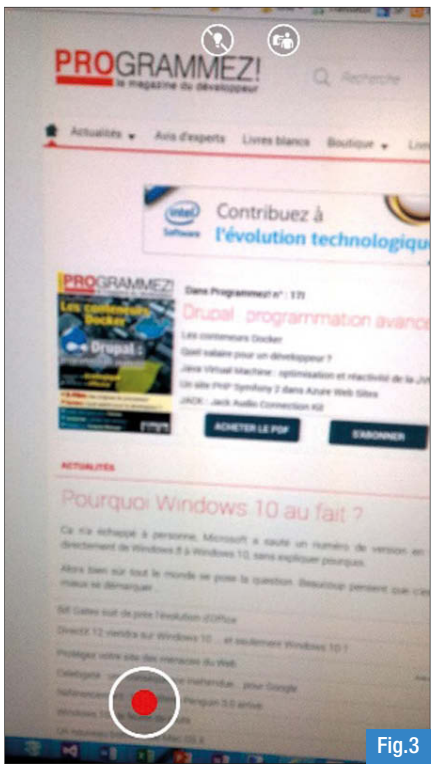


Fig.3

No Such Con 2014 : La force brute de la recherche !

La conférence internationale de hacking No Such Con s'est tenue du 19 au 21 novembre à l'Espace Oscar Niemeyer à Paris. Près de 400 participants ont assisté à un ensemble de présentations techniques de haut niveau, sur des sujets variés comme le hijacking d'accès Internet via CPL, la cryptographie quantique, ou encore le détournement de trafic via BGP. Cette édition hivernale de NoSuchCon offrait aussi la possibilité de se confronter à un challenge CTF (Capture The Flag) et pour les plus chanceux, places limitées oblige, un worshop de hacking hardware, fer à souder en main.

Véronique Loquet
Fondatrice de l'agence RP AL'X Communication. Spécialiste de l'Open Source et de la sécurité des SI.
sur Twitter @vloquet

Les intervenants sont tous des hackers, chercheurs, experts en sécurité, plutôt que plombiers de l'Internet. Ils sont venus de loin pour certains, afin de partager leurs travaux de recherche et quelques séquences inédites. Cette année il y avait aussi plus de chercheurs français qu'à l'habitude; ils n'auront cette fois pas attendu le SSTIC à Rennes pour partager leur savoir. Les actes complets de No Such Con sont disponibles ici : http://www.nosuchcon.org/talks/2014/
En attendant de vous lancer dans la recherche brute, voici un florilège du programme 2014.

A la mémoire de Grothendieck

C'est Rolf Rolles, expert en reverse engineering, qui a ouvert le menu avec un cocktail ultra technique de langage machine et de rétro-ingénierie. Cette présentation de haute volée technique portait sur l'automatisation d'un procédé de désobfuscation, basé sur le processus de 'Program Synthesis', dédié à la création de programmes dont on aura généré les règles de comportement. Rolf a démontré comment désobscurcir un code au niveau assembleur, dans le but d'aider à son analyse, notamment en éliminant les zones de texte jamais exécutées et en générant des règles à la volée. Sa présentation détaillée offre de bons exemples de code machine, de code assembleur et de code en représentation intermédiaire (IR). (http://goo.gl/NQgzGk)

Sans filet

Le bulgare Georgi Geshev a pris la suite avec une démo live, exploitant notamment la vulnérabilité des MoMs basés sur JMS (Java Message Services). Il a ainsi présenté divers cas d'intrusions et d'attaques *Man in the Middle.* Son exposé, *« Your Q is my Q »* portait sur la possible communication entre différentes applications, via l'utilisation des files d'attente et le détournement d'ActiveMQ, le MOM libre de la Fondation Apache.
http://www.nosuchcon.org/talks/2014/D1_02_Georgi_Geshev_Your_Q_is_my_Q.pdf

L'Internet du voisin dans ta prise électrique

On savait déjà comment pirater l'accès Wi-Fi de son prochain, mais Sébastien Dudek, chercheur à Sogeti, enfonce le clou avec le CPL (courant porteur en ligne) et surfe à l'oeil en empruntant la connexion du voisin.
Ce dernier se transforme en victime lorsqu'on y ajoute des attaques de *spoofing, snooping, poisoning* et encore bien d'autres affronts en *ing*.
Au départ, une faille dans le réseau électrique, puis des compteurs qui n'isolent pas le trafic. *« Les signaux CPL ne sont pas arrêtés par les compteurs électriques. Seuls les plus récents filtrent, mais les autres permettent de capter les signaux de tout un immeuble. »* nous dit Sébastien. C'est le cumul de failles qui permet l'accès au pirate, car le flux CPL est chiffré, mais nombre d'utilisateurs négligent d'activer la sécurité **Fig.1**.
La complaisance des sites comme *le bon coin* ou *eBay*, qui publient les photos des boitiers avec leur mot de passe unique *Direct Access Key (DAK)*, permettra de générer de nouvelles clés de chiffrement.
http://www.nosuchcon.org/talks/2014/D1_03_Sebastien_Dudek_HomePlug AV_PLC.pdf
A ce stade rappelons que nous sommes ici dans le cadre de la recherche avec pour objet l'amélioration de la sécurité des systèmes d'information. Pour le lambda qui souhaiterait s'adonner à cette forme de hijacking, rappelons ce que prévoit le code pénal : "entraver ou fausser le fonctionnement d'un système de traitement automatisé de données est puni de cinq ans d'emprisonnement et de 75.000 euros d'amende".
Au chapitre de la loi, gardons aussi en mémoire que chaque internaute est responsable de la sécurité de son accès à Internet. Cela soulève quelques questions et de quoi inquiéter papi Jean-René, plus expert au jardin que sur les réseaux, pourtant passible du délit pénal de défaut de sécurisation de sa connexion.

G-Jacking : code, infrastructure et sandbox !

Le développeur, quant à lui, doit aussi respecter les bonnes pratiques en matière de sécurité pour maintenir la fiabilité de ses applications hébergées dans le Cloud où moult pièges lui sont tendus.
C'est ce que Nicolas Collignon, CEO de Synacktiv, en charge du pôle pentest et R&D, est venu démontrer. Nicolas a présenté plusieurs vulnérabilités du service PaaS (Platform as a service) dédié à l'hebergement Cloud de Google pour des applications Web, GAE (Google App Engine). Ces applications développées en Python, Java, PHP et Go, et hébergées dans les datacenters de Google,

nécessitent une attention particulière du développeur.
On y apprend en outre que les mesures de protection de Google peuvent être contournées et ont même un effet boomerang.
http://www.nosuchcon.org/talks/2014/D2_03_Nicolas_Collignon_Google_Apps_Engine_Security.pdf

De la quincaillerie... au microcontrôleur

Le crochetage de serrure version moderne, c'est l'attaque du loquet électronique. Braden Thomas qui fut de longue date un disciple d'Apple, désormais Senior Research Scientist chez Accuvant, s'est attaché à "reverser" le contrôleur MSP430 des serrures électroniques. Il s'agit donc ici d'ouvrir les serrures de maisons dont la sûreté repose sur une infrastructure cryptographique unique pouvant être activée à distance via télécommande ou smartphone Android.

Laurent Bloch, chercheur émérite, nous éclaire : « *La protection de base de ce type de contrôleur repose souvent sur le claquage du fusible de protection du dispositif Joint Test Action Group (JTAG) qui contrôle l'accès au microprogramme. Braden Thomas s'est employé à contourner cette protection, ce qui lui a demandé des attaques physico-chimiques pour atteindre le contrôleur et sa mémoire EEPROM, puis l'utilisation du flash de son appareil photo pour effacer le contenu de cette mémoire. Après des mois de travail il a pu extraire le microprogramme, en effectuer la rétro-ingénierie et accéder à la clé secrète désirée.* » Cette technique qui laisse la porte ouverte à des cambriolages en masse et sans effraction, trouve heureusement son correctif puisque ce hacker éthique pratique la divulgation responsable des failles de sécurité. Il a donc transmis le fruit des résultats de sa recherche aux constructeurs afin qu'ils corrigent leur architecture. http://goo.gl/r0D2oq

La haute capacité créatrice des hackers n'est plus à démontrer, si Steve Jobs à ses débuts créait des boîtiers pour pirater les téléphones, de son côté, iTunes n'aurait peut-être même pas existé sans la plate-forme de téléchargement illégale Napster, qui a permis de familiariser le grand public au téléchargement de musique en ligne. Cette transition nous emmène faire un tour en Italie, à la rencontre d'Andréa Barissani.

Open Hardware et design à l'italienne !

Andrea est un chercheur en sécurité discret mais de renom international. Co-fondateur de

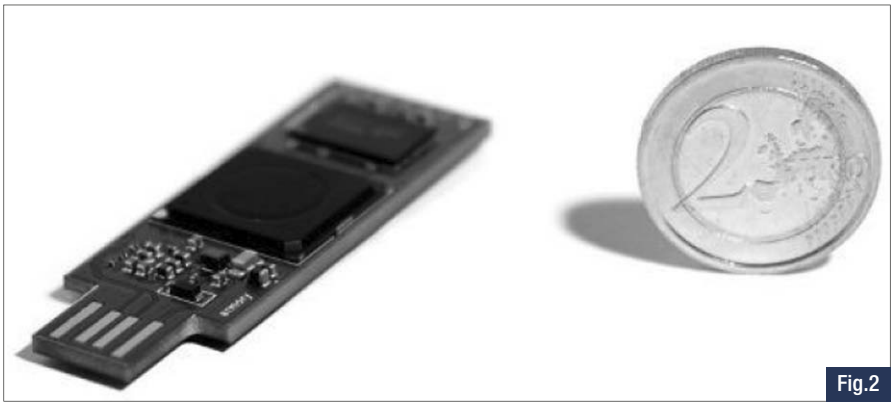


Fig.2

Inverse Path, il participe à de nombreuses conférences, avec une prédilection pour PacSec à Tokyo et CanSecWest à Vancouver, le plus souvent avec son alter ego Daniele Bianco. A NoSuchCon il est venu seul nous présenter la dernière release de USBArmory, prototype en poche à quelques semaines du lancement de la fabrication industrielle **Fig.2**.
USBArmory est présenté comme un couteau suisse USB.
C'est un ordinateur complet miniaturisé. L'objet de la conférence était de présenter l'ensemble des phases de construction du projet, de l'idée au produit **Fig.3 et 4**.
L'idée a germé en janvier de cette année, les schémas de développement ont pris forme en mars, le premier prototype a vu le jour en avril, s'ensuit une batterie de tests pour arriver à une version finalisée en novembre, prête pour la fabrication industrielle.
USBArmory sera commercialisé fin décembre, et déjà Andrea compte plus d'un millier de pré-commandes, tous les détails techniques sont consultables via
http://www.nosuchcon.org/talks/2014/D2_05_Andrea_Barisani_forging_the_usb_armory.pdf

Quand l'underground côtoie l'institutionnel

Guillaume Valadon et Nicolas Vivet, chercheurs à l'agence nationale de la sécurité des systèmes d'information (ANSSI), ont détaillé à l'aide d'exemples pratiques, la détection des attaques du protocole BGP. BGP est utilisé pour échanger des données de routage entre différents réseaux. Détournement de trafic, méthodologie des analyses en temps réel, contre-mesures et bonnes pratiques. Ils livrent une véritable antisèche pour les opérateurs !
http://www.nosuchcon.org/talks/2014/D3_04_Guillaume_Valadon_Nicolas_Vivet_detecting_BGP_hijacks.pdf

Capture the flag (CTF)

Le challenge développé par Nicolas Collignon et Eloi Vanderbeken de la société Synactiv, n'a révélé qu'un seul gagnant. Des défis ultra complexes relevés par Fabien Perigaud, expert en sécurité à Airbus, qui a su mettre à profit ses connaissances multiples, allant de la retro-ingénierie à la cryptographie. Lors de l'exposé des solutions Fabien explique la réalisation d'épreuve d'attaque de timing sur le cache du processeur pour exfiltrer une clé RSA.
http://www.nosuchcon.org/talks/2014/NSC_Challenge_intro.pdf
http://www.nosuchcon.org/talks/2014/NSC_Challenge_solution.pdf

Hardware Workshop

Mis en place par Damien Cauquil de Sysdream, cet atelier pratique a porté sur le hack d'un système de sonnette de porte sans-fil relié à une télécommande. L'idée était de démontrer d'une part, la facilité d'interception de signal de la télécommande, et d'autre part de définir un nouvel usage de cette télécommande. En l'espèce, un brouilleur de fréquences (interdit par la loi. Article L. 39-1) et un « bruteforceur », qui permettra d'émettre tous les codes en un temps record. C'est à ce moment de transformation qu'il devient nécessaire d'écouter l'instructeur et de s'armer d'un fer à souder et d'un peu (beaucoup) de dextérité **Fig.5**.

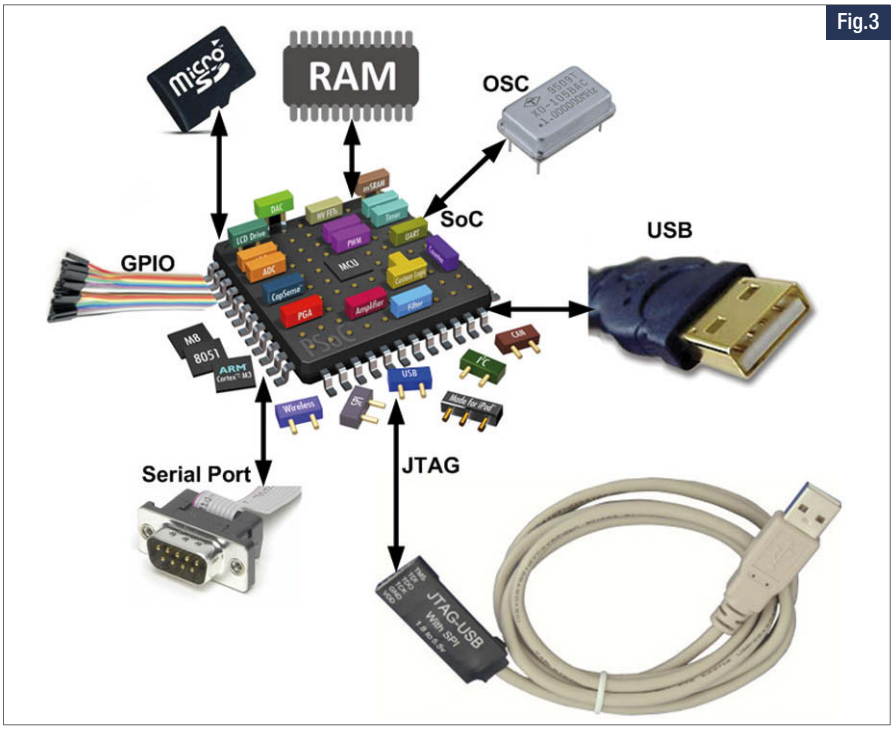
Impossible de citer ici l'ensemble des pwners au programme, Ezequiel Gutesman, venu démontrer les vulnérabilités d'HANA, le système de base de données en mémoire de SAP, Peter Hlavaty qui s'est s'attaqué au Kernel Windows, Andrea Allievi qui utilise Patchguard pour faire de l'injection de driver malicieux, Benjamin Delpy qui se joue des Patch Tuesday de Microsoft, avec son Mimikatz d'élévation des privilèges et open bar sur les comptes administrateurs...

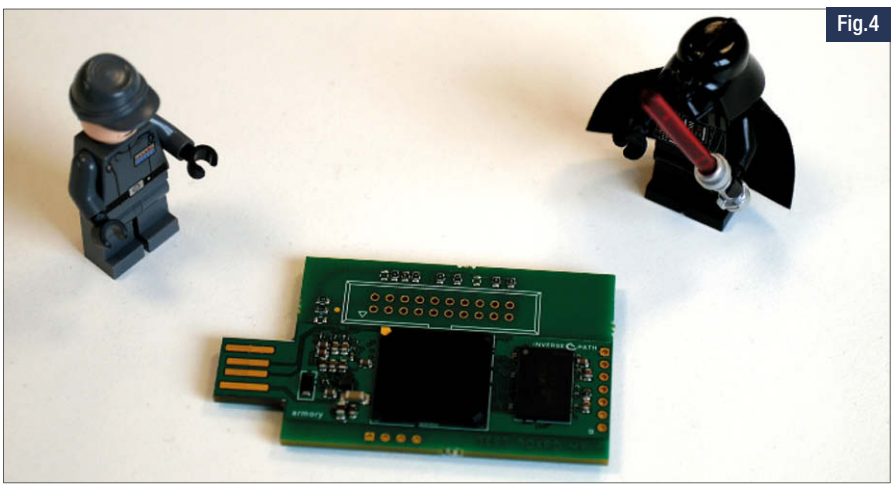
En point d'orgue... No Such conclusion, par-delà la technique

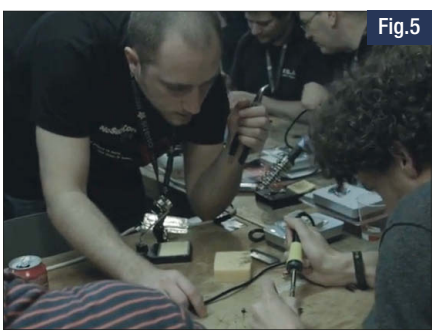
La keynote de clôture était assurée par Anthony Zboralski. Il cumule plus de 15 années d'expérience, et un blaze mythique « Frantic », qui a marqué l'histoire du hacking des décénies 80' 90'. Après une échappée de quelques années à Jakarta, il vit désormais à

Londres. Avec lui la conférence s'achève sur la reflexion. Il a vu évoluer ce milieu de la sécurité avec quelques mutations dans l'approche : « Au départ on regardait un système dans son ensemble, avec une méthode systémique, dans le cadre d'audits de sécurité par exemple. On n'était pas uniquement centré sur la cible, mais la cible dans son environnement, interne et externe. Aujourd'hui les gens ont développé des spécialités, ils sont moins en capacité d'observer et de tester globalement. » Il évoque les questions de l'échec de la sécurité illustré par des exemples concrets basés sur son expérience, pose la question de notre capacité à préserver le respect des libertés numériques et de la vie privée, l'absence de compréhension de nombre de dirigeants et souligne plus globalement la technophobie de nos technocrates.
La sécurité n'est pas uniquement une affaire de techniciens et l'enjeu n'est pas que celui des grandes entreprises et des États.
Outre les présentations techniques, cette conférence a toujours tenté d'ouvrir le débat sur l'équilibre social, l'accès aux savoirs, la démocratie numérique, le respect des données personnelles ou la formation et l'éducation. Sur l'apprentissage, en attendant le lancement promis d'un grand plan pour le numérique à l'école, Anthony Zboralski livre son point de vue : « Quand j'étais au Rotary Club de Jakarta je voulais faire une levée de Fonds pour apprendre aux gamins défavorisés à programmer et à hacker. Je prône l'apprentissage via une approche d'acquisition de la connaissance par le constructivisme en rupture avec l'instruction conventionnelle. Tenir compte de son environnement, développer un modèle mental et apprendre à développer de bons réflexes.
A l'image de l'initiative OLPC (One Laptop Per Child) de Negroponte Nicholas Negroponte, fondateur du MIT Media Lab]. Plus on est exposé tôt à un apprentissage, plus on a de chances de le maîtriser. »

Gageons que l'enseignement du code intervienne rapidement dans les programmes scolaires, et qu'on ne se limite pas à une multiplication d'annonces. L'Estonie fut un pays précurseur, d'autres lui ont emboité le pas comme le Royaume-Uni. L'informatique fait partie de la culture générale, elle est une science indispensable, son enseignement participera aussi à la compréhension de la sécurité.







Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € Autres pays : nous consulter.
PDF : 30 € (Monde Entier) souscription exclusivement sur www.programmez.com

Une publication **Nefer-IT**
7 avenue Roger Chambonnet
91220 Brétigny sur Orge
redaction@programmez.com
Tél. : 01 60 85 39 96

Directeur de la publication & rédacteur en chef : François Tonic

Ont collaboré à ce numéro : Sylvain Saurel

Secrétaire de rédaction : Olivier Pavie
Experts : E. Margraff, J. Fichet, P. Lorieul, A. Galtier, F. Bordage, J. Chatard, D. Rosset, T. Lebrun, C. Séjourné, B. Bonnet, X. Bucchiotty, P. Guillerm, D. Houssoun, C. Peugnet, T. Leriche-Dessirier, Michael Paris, R. Lifchitz, S. Grimonet, V. Loquet, S. Galand

Crédits couverture : Hack : © 09-25-14 © frankpeters
Igloo : 12-23-13 © Larry Rains, Intel, D. Rousset

Maquette : Pierre Sandré

Publicité : PC Presse,
Tél.: 01 74 70 16 30, Fax : 01 41 38 29 75
pub@programmez.com

Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.

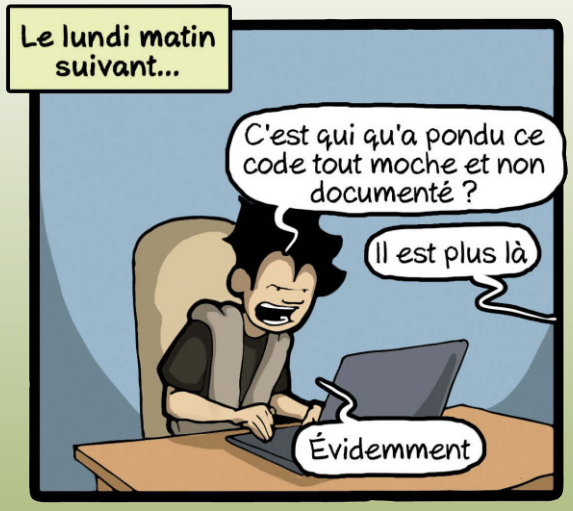
Marketing et promotion des ventes :
Agence BOCONSEIL - Analyse Media Etude

Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

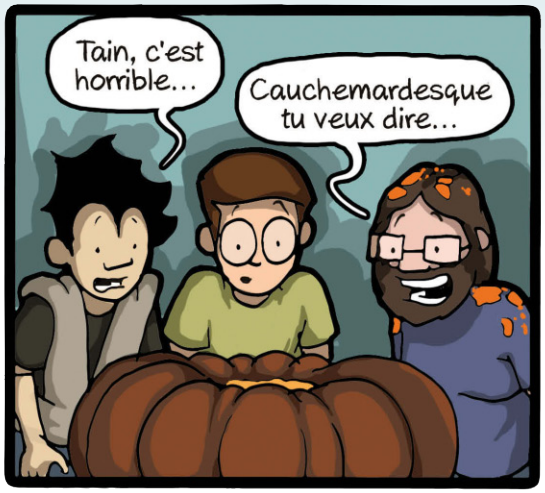
Contacts
Rédacteur en chef : ftonic@programmez.com
Rédaction : redaction@programmez.com
Webmaster : webmaster@programmez.com
Publicité : pub@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1215 K 78366 - ISSN : 1627-0908

Le départ d'Adrien



Halloween



Chaque semaine, de nouvelles aventures !

www.commitstrip.com

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

L'INFORMATICIEN
www.linformaticien.com