PROGRAMMEZ
PROGRAMMEZ!
le magazine du développeur
www.programmez.com

Développez sur Linux !

Votre PC de développement idéal

Conférence //BUILD 2014

Open Source, Azure, Visual Studio...
Les Applications Universelles Windows

Java 8

La révolution des lambdas

Langage fonctionnel

Késako ?

iOS

A la découverte d'Objective-C

Web
(re)découvrez Grails

Antiquité
L'histoire d'Eclipse

Design
Prototyper une application

Impression 3D
La 3D FreeSculpt

Mensuel n°174 - Mai 2014
M 04319 - 174 - 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

LINUX Solutions Libres & Open Source

Le salon dédié à linux et aux logiciels libres

20&21 MAI 2014

CNIT - Paris La Défense

Toutes les solutions et nouveautés informatiques en Open Source...
Pour encore plus de libre au service de l'entreprise !

Un événement Tarsus FRANCE GROUPE MÉDIA B TO B

Partenaire officiel monANNUAIRE pro.com

www.solutionslinux.fr

édito

// mode : sauve qui peut

Plus que fort le Goto, voilà Heartbleed

Décidément depuis janvier, les failles critiques se succèdent. Nous avions eu l'incompréhensible erreur de programmation Goto dans iOS, puis la superbe faille dans GnuTLS, c'est au tour du protocole SSL de faire peur à tout le monde et pas seulement aux geeks.

« La faille informatique, nommée « Heartbleed », a été identifiée au sein de la bibliothèque logicielle OpenSSL. Cette dernière, en créant des tunnels chiffrés selon les protocoles SSL ou TLS, est conçue pour apporter confidentialité et intégrité aux échanges effectués avec de nombreux services sur Internet : sites Web transactionnels (HTTPS), serveurs de messagerie (SMIME), accès distants aux entreprises (VPN SSL), etc. Ainsi, l'exploitation de cette vulnérabilité permet de récupérer à distance des données présentes dans la mémoire du serveur impacté. Les données collectées peuvent être les clefs de chiffrement associées au protocole de sécurité, des identifiants et des mots de passe d'utilisateurs du service, ou toute autre donnée transmise par l'utilisateur. » commentaient les experts de Lexsi. La faille permet de récupérer tout ou partie des données utilisateurs, selon les services et les scenarii. Des milliers de sites étaient donc vulnérables, à des degrés divers.

Heureusement, les communautés ont réagi rapidement pour créer des correctifs et permettre aux webmasters et développeurs de patcher les serveurs...

Cela démontre une fois de plus que la sécurité informatique, comme les tests logiciels, ne doit pas être juste un mot, ou encore, une variable d'ajustement. Quand à Programmez ! nous répétons année par année que la sécurité passe aussi par le développeur, la programmation sécurisée, ce n'est pas pour vous « emmerder ».

Il y a 20 ans, en qualité de testeur logiciel, j'avais la charge de vérifier que tous les champs de saisie et d'affichage étaient strictement bornés avec les critères définis dans le cahier des charges. Une faille dans un masque de saisie ou un filtre, et c'était potentiellement le risque d'exploser une requête, de corrompre une table ou une base de données entière. Quand je vois les dysfonctionnements, parfois élémentaires de sites web, d'applications, je me dis : en 20 ans, nous n'avons rien compris, rien appris.

Ces failles, et négligences, s'amplifient avec les applications mobiles, les objets connectés et les données qui se transmettent partout, sans savoir exactement comment et pourquoi.

\\ on base
\\ fundamental, on ne peut-être sûr de rien...

0a26df6e372fb9c720ec75663a2fc28e (*)

(*) la lectrice ou le lecteur qui donne la première bonne réponse gagne une clé USB Programmez ! : qu'est-ce qui se cache derrière cette chaîne et le type de cryptographie utilisée. Réponse : redaction@programmez.com

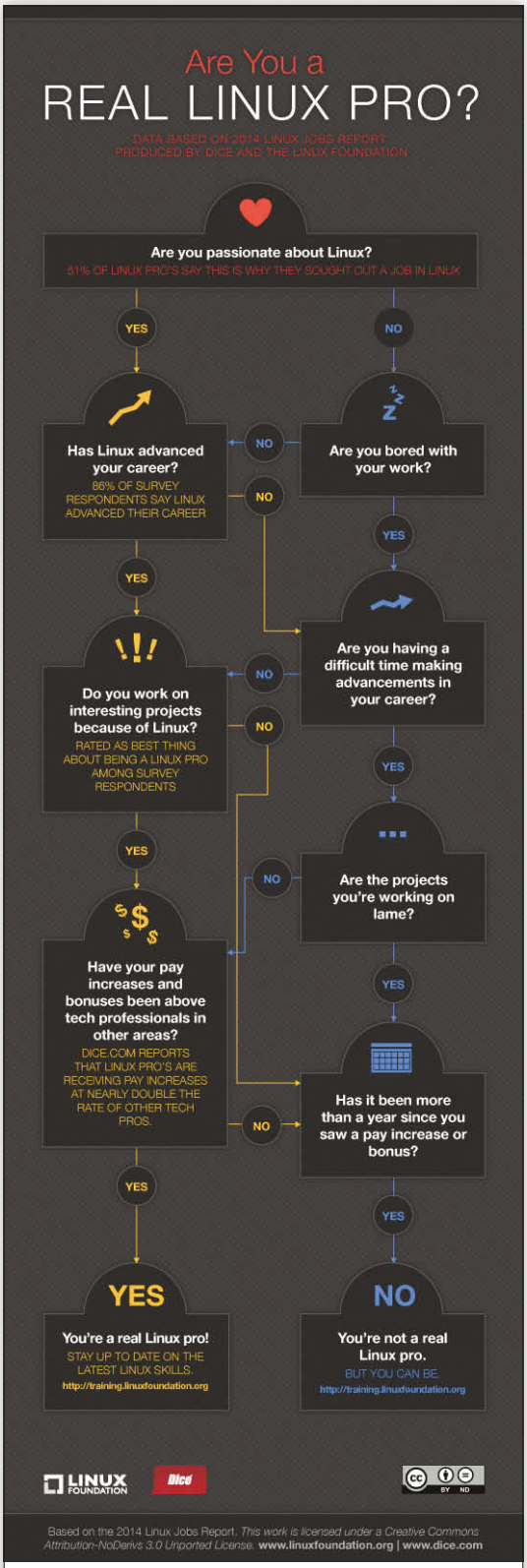
sommaire

À LIRE DANS LE PROCHAIN NUMÉRO
n° 175 en kiosque le 30 mai 2014

Créer et gérer son projet open source

Spécial développement mobile
Firefox OS, Windows Phone 8.1, Nokia X, Xamarin...

Devoxx France 2014
reportage complet



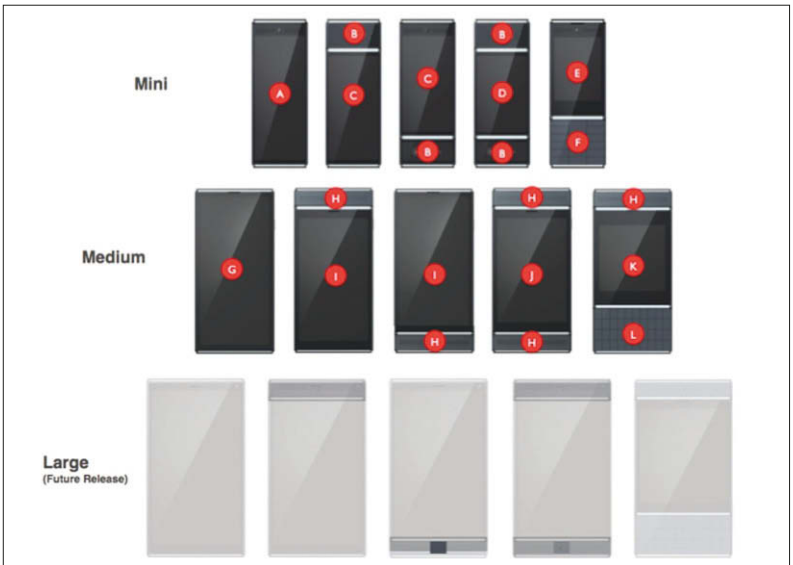
Compétences Linux : les priorités 2014

- *trouver des compétences Linux*
- *recrutement en hausse de spécialistes Linux dans les 6 mois*
- *opportunité de carrière : bonnes perspectives !*

La fondation Linux et Dice ont dévoilé leur nouveau rapport sur l'emploi dans le monde Linux. Selon les rapports, de plus en plus d'entreprises cherchent des compétences Linux. Le recrutement de compétences Linux devrait continuer à croître en 2014, jusqu'à +70 %. La sécurité reste un des moteurs de ces recrutements.
3 mesures incitatives souvent fournies aux experts Linux :

- *horaires et organisation plus souples*
- *hausse des salaires*
- *certification et formation*

Mais Linux continue à être un travail et une passion. Surtout, désormais, les compétences Linux sont des atouts pour une carrière, trouver un poste, évoluer.



Ara : le téléphone comme vous le montez

Avec le projet Ara, Google veut proposer un téléphone modulaire, à monter soi-même. Sur une base standard, vous pourrez rajouter les modules désirés, changer la configuration très rapidement et en cas de panne d'un élément, le changer tout aussi facilement. Trois tailles sont prévues : mini, moyen et grand. Original et totalement geek ! Un kit de développement est disponible...

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

Avril 2014	Avril 2013	Tendance	Langage	%	Evolution (en %)
1	1	↓	C	17,631	-0,23
2	2	↓	Java	17,348	-0,33
3	4	↑	Objective-C	12,875	+3,28
4	3	↓	C++	6,137	-3,58
5	5	↓	C#	4,820	-1,33
6	7	↓	(Visual) Basic	3,441	-0,126
7	6	↓	PHP	2,773	-2,65
8	8	↓	Python	1,993	-2,45
9	11	↑	JavaScript	1,750	+0,24
10	12	↑	Visual Basic .NET	1,1748	+0,65

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. Le trio de tête reste identique : C, Java et Objective-C. Ce dernier connaît une belle petite progression. La plupart des autres langages subissent une baisse, exceptés VB et JavaScript.

POURQUOI CHOISIR MARIADB ?

Dans la guerre des bases de données, je demande MariaDB 10 contre MySQL. Avec la disponibilité de MariaDB 10, SkySQL veut frapper vite et très fort !

Fonctions	MariaDB 10	MySQL
réplication multi-source	oui	non
amélioration du partionnement des tables	oui	oui
moteur de stockage TokuDB	oui	nécessité un module tiers
InnoDB : amélioration du moteur	oui	oui
Fusion-io : fonctions spécifiques	oui	non
optimisation des sous-requêtes	oui	non
outil CONNECT pour connecter les moteurs de stockage	oui	non
Cassandra : moteur NoSQL	oui	non
Dynamic Column : pour le stockage et traitement similaire à du NoSQL	oui	non
interface memcached NoSQL	non	oui
fonctions et outils orientés DevOps	oui	partiel

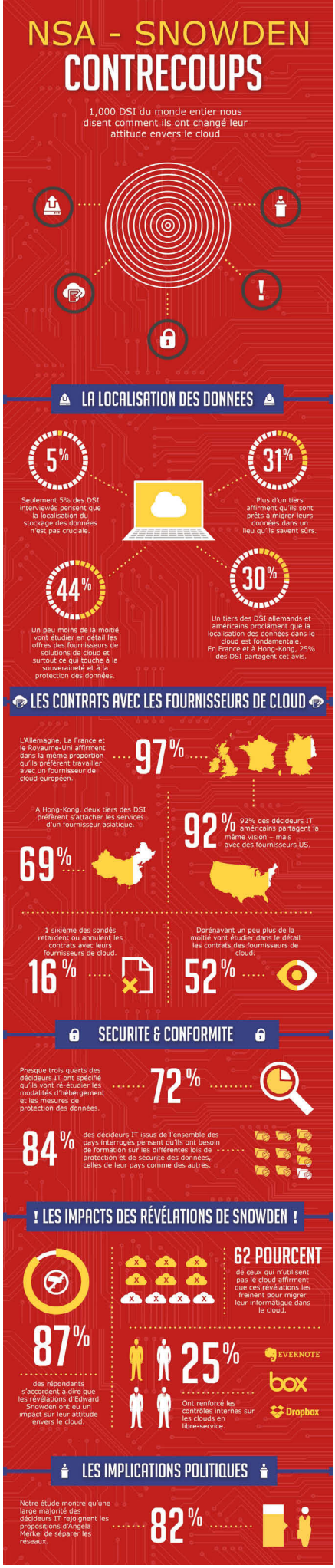
(source : SkySQL / extrait de la comparaison publiée par SkySQL)
Promis, MySQL 5.x comblera les retards fonctionnels...

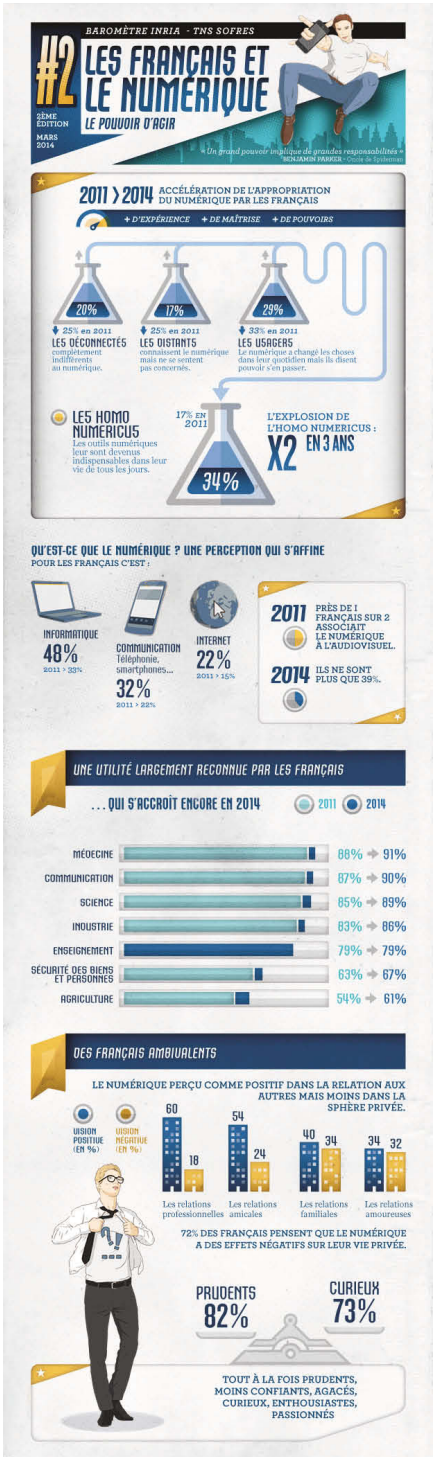
4G

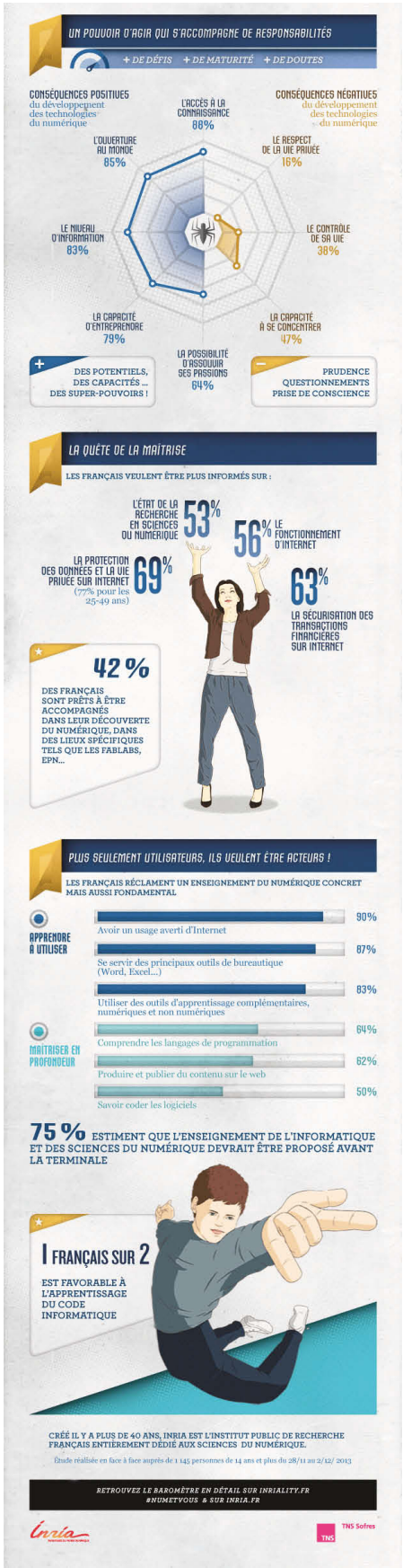
la Surface 4G est enfin disponible

Open Data :

priorité du gouvernement ? Pas certain.





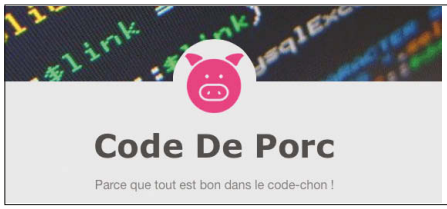


Amazon

devrait lancer ses propres téléphones dans les prochains mois

Thunderbolt

placez les périphériques en début de chaine pour un maximum de performances.



Du code cochon (version maxi-hardcore) ?

Une seule adresse à consulter d'urgence (âme de développeur sensible, passez votre chemin, cela fait mal, très mal) :
http://code-de-porc.tumblr.com

Challenge hacking Europe – Afrique : HNC

L'association ACISSI a lancé son 6e challenge Hacknowledge Contest 2014-2015. Ce challenge de sécurité informatique se déroulera sur deux années. La première étape a eu lieu fin avril dernier en Belgique, d'autres suivront. Chaque « rencontre » se déroule sur 12 heures et met à défi plusieurs équipes. Plusieurs challenges sont proposés, sur différents thèmes. En tout, + 60 épreuves sont déjà prévues (forensic, matériel, réseau, systèmes industriels, etc.).
Pour en savoir plus : http://www.hacknowledge-contest.org

mai

Android Day, 15 mai

L'école d'informatique ESGI accueille Android durant une après-midi entière. L'agenda sera le suivant :

- Initiation à Android
- Push Notification
- Nokia X
- Mobile Wallet sous Android
- Géolocalisaiton
- Google Glass...

La conférence s'articulera sur des conférences et des ateliers techniques.
Entrée libre sur inscription auprès de Stéphanie Petit (spetit@esgi.fr)

Ncrafts, 16 mai

Une conférence indépendante sur les technologies et langages .Net, voilà tout le défi de la journée NCrafts. Les organisateurs veulent faire un panorama technique assez complet de la plate-forme : Elasticsearch, Roslyn, F# et meta-programmation, DDD, basse latence système, refactoring. Une belle initiative ! Venez nombreux. site : http://ncrafts.io

dotScale, 19 mai

La conférence européenne dotScale revient. Elle se concentre sur le Cloud et le Big Data. Plus de 700 personnes sont attendues ! La conférence est dédiée aux développeurs. DNS, Netflix, Google, Rackspace, OpenZFS seront présents durant cette journée événement. « Le but de cette journée pour les développeurs ? Comprendre comment architecturer leurs applications pour les rendre « scalable », c'est-à-dire capables de gérer des pics de trafic très importants. En complément de la conférence, des ateliers gratuits et ouverts à tous seront organisés par les partenaires le 17 mai », précise l'organisation. Tarif normal : 129 €. Toutes les sessions se feront uniquement en Anglais. Site : dotscale.eu

Solutions Linux, Libres et Open source

Le grand salon Linux et Open source aura lieu les 20 et 21 mai à Paris – La Défense. Cette année, 9 thématiques seront présentes :

- Cloud Libre
- Open source en entreprise : consommation, maîtrise, mutualisation,
- Les nouveaux enjeux des outils collaboratifs Open Source,
- BI / Big Data,
- L'Open Source au cœur de la révolution mobile,
- Développement web
- CMS, CMF, développement spécifique
- Sécurité
- Administration Système / clusters / DEVOPS

site web : http://www.solutionslinux.fr

Conférences des JUG

Toulouse Jug : soirée le 15 mai (non confirmée).
Site : http://toulousejug.org
Jug Lyon : Java 8 (à confirmer). Site : http://www.lyonjug.org
Jug Bordeaux : soirée DART le 22 mai.
Web & sémantique le 11 juin.
Site : http://bordeauxjug.org/meetings
Riviera JUG – Sophia-Antipolis : soirée Lambda, sex and sun... 13 mai. Site : http://rivierajug.org

juin

EclipseCon France 2014

Les 18 et 19 juin, la conférence Eclipse France aura lieu à Toulouse. L'appel aux contributions s'est déroulé en avril et l'agenda sera dévoilé courant mai. Plusieurs sessions sont déjà prévues : CoffeeScript, Mathus, débug et tracing tool. EclipseCon France est la conférence officielle de la fondation. Site : https://www.eclipsecon.org

PHP Tour Lyon 2014

Cette année, l'AFUP arrête la caravane du PHP Tour à Lyon les 23 et 24 juin prochains. Sur 2 jours, vous pourrez rencontrer les acteurs de communautés PHP en France et quelques invités surprises.
De nombreuses sessions techniques seront proposées. Le programme sera disponible très bientôt.
Site : http://afup.org/pages/phptourlyon2014/index.php

Une conférence, un événement pour développeur ?
Envoyez-nous votre agenda : redaction@programmez.com

1&1 DOMAINES

0,99 €*

Votre nom de domaine .fr .com

- **Inclus avec chaque nom de domaine :** redirections, gestion complète du domaine, création de sous-domaines
- **Support expert** 7j/7, 24h/24 via hotline non surtaxée et email

1&1

MEMBRE DE united internet

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

0970 808 911 (appel non surtaxé)

1and1.fr

* Noms de domaine .fr et .com : 1 an à 0,99 € HT (1,19 € TTC). À l'issue de la 1ère année, le prix habituel s'applique. Offre à durée limitée et soumise à conditions détaillées disponibles sur 1and1.fr.

Chapitre 7 : DIY, DIT et Connected Objects, révolution du consumérisme ?

Nous voici déjà au chapitre 7 de cette épopée au sein de la Silicon Valley. De nombreux sujets phares y ont été abordés, comme l'impression 3D, l'intelligence artificielle, la robotique, la voiture électrique ou encore des nouveaux eldorados comme celui de la data et des objets connectés. Mais quel est le point commun de tous ces domaines ? Et y en a-t-il d'autres ?

Quel est ce mouvement engagé par les « Makers », le « Do It Yourself » (DIY que nous pouvons traduire par « faites-le vous-même ») qui donna naissance à de nombreux manifestos du « Repair by Yourself » avec des marques, tous domaines confondus, comme Make:, iFixit, Do it Yourself, Sugru et bien d'autres encore que nous découvrirons au long de cet article.

L'utilisateur est-il arrivé à un stade où, à trop vivre l'ultra-consumérisme, il souhaite maintenant se réapproprier les objets qui l'entourent, retrouver le droit de penser, d'agir et non plus de laisser la télévision, émissions ou médias quelconques injectés de publicités penser pour lui, ou peut-être souhaite-t-il tout simplement faire, juste faire par lui-même ou ensemble !

Repair by Yourself ou Repair Together !

Rappelez-vous ces dimanches merveilleux chez vos grands-parents à bricoler en compagnie de votre bon papa occupé à réparer la machine à coudre de votre grand-mère, le moteur de voiture alors accessible pour le commun des mortels, ou encore de bricoler les premiers ordinateurs affichant un bel écran noir, vert ou orange doté d'un emblématique « C:\> ». Souvenirs !

Le « Repair by Yourselft », bien que commun pour les générations nous ayant précédé, semble être redécouvert par notre société. Les groupements ou sites internet du Do It Yourself fleurissent en faisant la promotion de leur différents manifestos. Voici quelques exemples de manifestos du Repair by Yourself :

iFixit : http://www.ifixit.com/Manifesto

Platform 21 : http://www.platform21.nl/page/4375/en

Sugru : http://sugru.com/manifesto

Leur point commun étant une revendication au nom des utilisateurs au droit de disposer de la liberté de réparer tout objet et de refuser l'obsolescence préprogrammée.

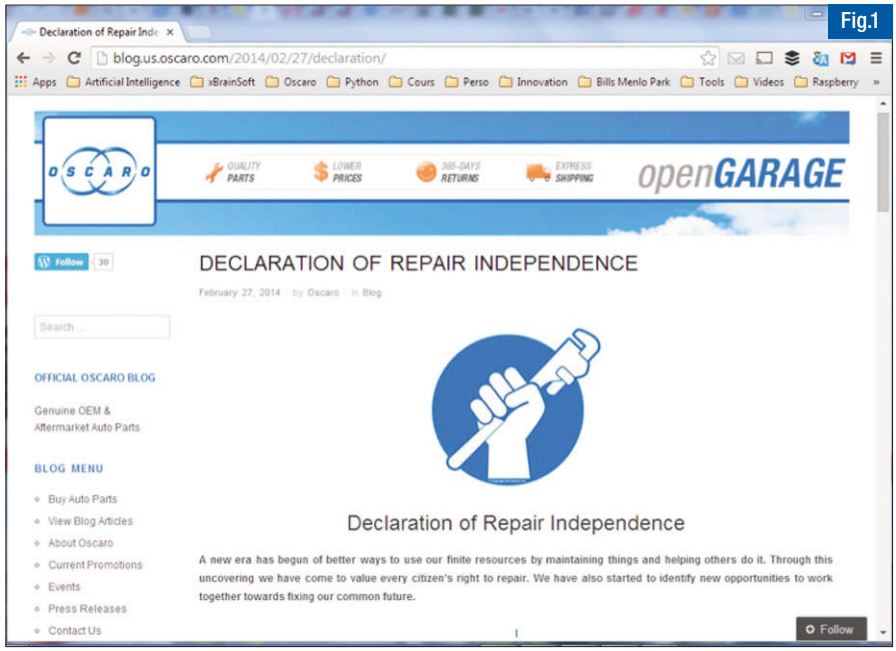
Prenons l'exemple de **iFixit** dont la mission est de vous aider à réparer l'ensemble de vos objets : mobiles, tablettes, ordinateur, mac, appareils ménagers ou encore voitures... iFixit vous propose plusieurs milliers de modes d'emplois pour vous aider à prolonger la durée de vie de vos objets préférés.

Mais où est le business modèle ? Non content de vous apporter ces modes d'emplois, Saint Graal de la protection de votre porte-monnaie, iFixit vous propose un modèle se différenciant via la vente de composants, ainsi que tous les outils nécessaires afin de vous permettre d'atteindre votre objectif ultime : « Que ce maudit smartphone puisse fonctionner à nouveau sans que je doive débourser une somme affolante pour y arriver ».

Le modèle existant se voit ainsi transformé d'une consommation poussée à l'extrême en un modèle plus ouvert et collaboratif avec un changement des typologies de dépenses créant une rupture des modèles courants.

Et ils ne sont pas seuls, nombreuses sont les startups au sein de la Silicon Valley engageant le pas de ce nouvel eldorado du « Do It Yourself » ! De nombreux sites e-Commerce prédestinés à ce type de mouvement embrassent ainsi très justement cette démarche communautaire en poussant les utilisateurs vers l'autonomie et l'indépendance réparatrice Fig.1.

Bien que ce type de mouvement ne soit pas neuf, voire très connu dans le domaine du soft-



L'Open Garage et la déclaration d'indépendance de réparation de votre véhicule – Manifesto d'Oscaro.com.

ware avec les communautés de développeurs online ou groupes utilisateurs depuis plusieurs décenniese celui-ci se généralise à l'ensemble des domaines de notre vie.

Cette tendance s'organise de plus en plus avec les Makers Faire dont l'origine se trouve en Californie à San Mateo. Créé en 2006, le Maker Faire Bay Area a célébré son huitième anniversaire en 2013 avec la visite de plus de 120 000 personnes.

Alors que le Maker Faire est de plus en plus populaire, de nouveaux évènements ont vu le jour à travers le monde (New-York, Detroit, Paris, Montreal, ...) : http://makerfaire.com/ **Fig.2.**

Créés par la communauté et gérés de façon indépendante, les Mini Maker Faire sont inspirés de leurs grands frères et sont présents dans plusieurs villes dont, en France, à Saint Malo.

Maker Faires around the World : http://makerfaire.com/map/

Ainsi le mouvement du Maker Faire ouvre spontanément la démarche du DIY à une approche plus collaborative donnant naissance au **Do It Together** (DIT).

N'est-il pas dommage de devoir passer par toutes ces étapes marketing pour revenir au bon sens et au fait que tout un chacun était, il y a encore peu, un « Geek » et non pas seulement en informatique ? Peut-être est-ce le seul moyen de passer d'une économie capitalistique à une économie sociale et solidaire. Le Maker Faire devenant ainsi le petit village d'antan !

Du DIY aux Objets Connectés

Après avoir gouté au DIY ou le DIT, en bon Geek, il ne vous faudra pas longtemps pour passer au souhait de vouloir connecter tout votre environnement. Qui n'a pas rêvé d'une voiture type Kitt, d'une maison pilotée par Sarah, ou encore de transports en commun accompagnés par Shirka.

Le monde des objets connectés vous ouvre les bras, alors lancez-vous sans plus attendre. Une imprimante 3D à 299 dollars (https://www.kickstarter.com/projects/m3d/the-micro-the-first-truly-consumer-3d-printer) et un Raspberry Pi à 35 dollars (http://www.raspberrypi.org) agrémenté d'un Arduino à moins de 30 dollars (http://www.arduino.cc) vous suffiront pour démarrer la construction de votre propre objet connecté **Fig.3**.

A vous créativité, connectivité et objets connectés! A vous l'envie de créer votre propre startup prête pour ce nouvel Internet, l'Internet des Objets (IDO), à la croisée des chemins des différents web : Social, Physique, Sémantique, Temps réel et Programmable. (http://fr.wikipedia.org/wiki/Web_des_Objets)

Ainsi l'Internet des objets (IDO) ou Internet of Things (IOT) peut être désigné comme étant l'expansion d'Internet à des choses ou objets de la vie quotidienne et à des lieux dans le monde réel.

L'Internet des objets désigne pour sa part le « mouvement de liaison » d'objets de votre quotidien ou de nouveaux objets à Internet. Souvenez-vous des dernières annonces de rachat de Google ou Facebook, pour ne citer que ceux-ci. Robots, voitures, vélos, ballons, raquettes, bracelets d'activité ou de santé, thermostats, scanners pour faire vos courses à la maison, brosses-à-dents, lunettes et bien plus encore !

Eh oui, bienvenue dans le monde des objets de la Silicon Valley, tout sera tôt ou tard connecté au Cerveau Global que vous le vouliez ou non. Tout sera à disposition de la Artificial General Intelligence (AGI) poursuivie par des acteurs fortement implantés sur Menlo Park, Mountain View ou Cupertino.

Ces objets sont tous caractérisés par une identification, la gestion du contexte, une sensibilité à son environnement, une capacité d'interaction réactive ou proactive, voire de l'autonomie, ainsi qu'une représentation virtuelle au travers de votre smartphone.

Ces objets sont une forme de résultante des différentes lois de Moore (moins coûteux et plus puissants), de Metcalf (l'utilité d'un réseau est proportionnelle au carré du nombre de ses utilisateurs., soit, en clair, plus il y a d'utilisateurs dans un réseau plus celui-ci a de la valeur) et le Big Data, nouvel or noir dont nous avons suffisamment parlé dans les articles précédents. Les études varient et annoncent pour certaines entre 30 à 80 milliards d'objets connectés à horizon 2020. Peu importe le nombre à la fin, le marché potentiel est exponentiel et chacun peut y trouver sa place. Il vous faudra cependant rester vigilant car 2014 est encore le tout début de ce nouveau marché et il n'est pas rare de voir des objets pourtant bien sympathiques disparaître rapidement de par une non cohérence avec les attentes des consommateurs. Il suffira de se rappeler quelques aventures françaises comme le Nabaztag.

Fig.3

Quels enjeux, quel avenir ?

Alors, quel est le point commun entre l'Internet des objets, du Raspberry Pi ou du Arduino, les énergies renouvelables, les logiciels libres, l'économie sociale et solidaire, l'intelligence artificielle ou encore les imprimantes 3D ? L'essayiste américain Jeremy Rifkin identifie tous ces phénomènes dans une convergence unique de transformation de notre économie mondiale. Rifkin envisage une société où le capitalisme sera éclipsé par des mouvements collaboratifs et la production à petite échelle réduisant à néant les frais de production, stockage et de distribution. Très loin des modèles actuels.

Le capitalisme, synonyme d'investissements massifs rémunérés par le gain d'économies d'échelle, se voit mis à mal avec le coût marginal (coût à produire une unité supplémentaire) réduit à néant, ces technologies propres au Web des objets et du Do It Yourself changent complètement la donne.

Je vous invite à visionner une présentation de Rifkin sur le sujet : http://youtu.be/ZjrbGmdT8Qc

Liens

http://en.wikipedia.org/wiki/Do_it_yourself
http://connected-objects.fr/
http://fr.wikipedia.org/wiki/Web_des_Objets
http://www.objetconnecte.net/histoire-definitions-objet-connecte
http://fr.wikipedia.org/wiki/Loi_de_Metcalfe
http://en.wikipedia.org/wiki/Artificial_general_intelligence
http://www.cio-online.com/actualites/lire-l-internet-des-choses-passera-de-15-a-80-milliards-d-objets-connectes-entre-2012-et-2020-5302.html

Gregory **Renard** (Redo)
Geek in Silicon Valley
http://gregoryrenard.wordpress.com

Testeur : un beau métier…

Son nom est Benjamin Roux. Il est Software Development Engineer in Test chez Microsoft à Seattle. Comme son titre l'indique, il travaille dans l'équipe de test.

« Ma formation est plutôt basique, après le bac j'ai enchainé sur un DUT Informatique à l'IUT de Clermont-Ferrand, suite à ça j'ai intégré SUPINFO pour décrocher un Master of Science. Je travaille actuellement dans la division WinCXE de Microsoft (Windows Customer Experience) et, plus spécifiquement, je suis dans l'équipe Graphics où je m'occupe de GDI/GDI+ pour Windows 8. Début Avril je vais intégrer l'équipe Apps – toujours dans WinCXE – en tant que développeur. Dans cette nouvelle position je devrai corriger les bugs que l'on peut trouver dans les applications « inbox » de Windows 8 mais aussi dans les applications « legacy » comme Paint ou la calculette», précise notre testeur.
De manière générale, Windows CXE s'occupe de délivrer toutes les mises-à-jour que vous recevez via Windows Update. Le travail accompli ainsi que l'impact sont assez impressionnants. Les bugs que nous résolvons varient du bug graphique reporté par des utilisateurs sur des forums, à des bugs de sécurité 0-day.
Avant de venir chez Microsoft, il a travaillé pendant 4 ans à Montréal en tant que développeur/consultant dans une entreprise spécialisée dans les technologies Microsoft.

Comment es-tu tombé dans l'informatique et plus spécialement dans le développement ?
Jusqu'à ma dernière année au lycée, mon parcours post-bac était tout tracé, je voulais faire Math Sup pour ensuite devenir professeur de mathématiques, puis j'ai fait la connaissance d'un camarade qui était à fond dans le développement sur calculatrice (Texas Instruments) et c'est là que j'ai eu le déclic. J'ai donc commencé le développement en utilisant le Visual Basic des Texas Instruments. Suite à ça, j'ai changé d'idée de cursus pour me diriger vers un DUT Informatique – chose que je ne regrette pas. Depuis ce temps-là, ma passion n'a fait que grandir et je suis persuadé d'avoir fait le bon choix !

Tu as gardé un regard très geek : gadget, veille techno. C'est important pour ton job et ta passion ?
Très important. La première chose que je fais quand j'arrive au travail, c'est de regarder les flux RSS (oui je suis has-been sur ça) des sites d'actualité informatique US, et, dans une moindre mesure, Français. Je suis le premier à acheter des gadgets inutiles mais j'adore ça. Bizarrement nous sommes une minorité chez Microsoft, beaucoup de personnes changent complètement de vie après le travail.

Etre développeur n'est pas toujours facile : pression, évolution constante, frustration des projets et des «chefs», c'est quoi pour toi être développeur aujourd'hui ? Le job a-t-il changé depuis tes débuts ?
Je vais parler du travail chez Microsoft, qui est très différent de ce qui se fait ailleurs. Ici c'est sans stress, les horaires sont libres et il n'y a personne qui joue au « chef » sans bonne raison – tout du moins pas dans ma team/division. Chez Microsoft le développeur est roi. En revanche lorsque je travaillais à Montréal – plus dans l'esprit Français déjà – alors oui certaines choses étaient frustrantes. Les « chefs » non techniques qui ne comprennent pas les contraintes techniques de la plateforme imposée ou les commerciaux qui vendent des fonctionnalités qui n'existent pas et qui nous le jettent à la figure en rentrant de rendez-vous par exemple. Pour moi le job a changé tout simplement parce que j'ai changé de compagnie, mais bon je ne peux pas généraliser avec ma petite expérience de 5 ans. Je ne ressens pas non plus de pression en étant dans Microsoft, bien que notre travail soit très critique, les délais donnés sont très respectables et permettent que tout soit prêt dans les temps. Dans le cas d'un retard pour telle ou telle raison, il n'y a aucun problème pour retarder la mise à jour pour le mois suivant – à condition qu'elle ne soit pas critique bien entendu.

Et en dehors du boulot, qu'est-ce que tu aimes faire ? Comment trouves-tu l'équilibre entre travail, vie privée, passion, famille ?
Ca va faire très cliché, mais l'ordinateur fait partie intégrante de mon après-travail. Ma fiancée finit souvent tard – elle travaille dans la plus grande société de gestion d'investissement au monde – du coup j'ai pas mal d'heures « libres » où je peux jouer et surtout développer mes propres applications pour Windows Phone ou Windows 8. Comme beaucoup, je regarde aussi énormément (trop ?) de séries et cela me prend un temps considérable – du temps bien dépensé je dirais. En dehors des écrans je passe beaucoup de temps à jouer avec mon chien – un Border Collie – qui a besoin de beaucoup d'exercice. Le week-end par exemple c'est direction les pistes de « randonnées » pour que Yuna puisse se défouler. C'est une des choses que j'adore dans la région de Seattle, tout est à proximité. Que ce soit la plage, la montagne,

Montée en charge linéaire et extrêmement performante

NCache™
100% .NET

Pour applications .NET et Java
(supporté sur Windows Azure et Amazon AWS)

JvCache™
100% Java

Les données en cache (via NCache), réduisent les accès coûteux en base, et permettent à vos applications de monter en puissance vers “extreme transaction processing” (XTP). JvCache est une implémentation native 100% Java de NCache.

Cache distribué en mémoire

- Extrêmement rapide et montée en charge linéaire avec 100% uptime
- Topologie en Miroir, Répliquée, Partitionnée et Cache Client
- NHibernate et Entity Framework cache niveau 2

Optimisation ASP.NET de Web Farms

- ASP.NET Session State cache
- ASP.NET View State cache
- ASP.NET Output Cache provider

Memcached Wrapper Fourni

Partage de données en mode Runtime

- Notifications d'événements puissants pour le partage de pub / sub données

Alachisoft
sales@alachisoft.com
US: +1 (925) 236 3830
Siège de l'entreprise

Télécharger un essai GRATUIT!
www.alachisoft.com

RedFabriQ
info@redfabriq.com
Tel: +33 1 40 16 07 89
Distributeur et intégrateur en France

les pistes de skis, les pistes de randonnées, tout peut être atteint très rapidement. Et finalement, quelques jours après mon arrivée à Seattle, mon camarade Julien m'a fait découvrir le wakeboard que j'ai plutôt bien aimé, donc j'attends avec impatience d'y retourner (la saison a déjà recommencé pour lui !). Sur un autre registre, j'aime aussi tout ce qui est voiture, notamment les voitures de sport mais aussi prendre mon Jeep pour aller faire du hors-piste dans la région.

Peux-tu nous présenter ton quotidien en quelques mots ?

Ici je vais parler de mon quotidien en tant que testeur puisque je n'ai pas encore commencé mon nouveau poste. Alors premièrement, le rôle de testeur ici n'a rien à voir avec le poste de QA que l'on peut trouver en France. Le poste de testeur – tout du moins dans ma division – est très technique et le processus de recrutement est le même que pour les développeurs.

Mon quotidien consiste donc à participer à la création de mises à jour pour les composants Windows dont mon équipe s'occupe. Le processus se déroule de la manière suivante :

- Un bug est signalé par des utilisateurs ou des partenaires,
- Le développeur et le testeur cherchent la cause du problème dans le code,
- Le développeur s'occupe de coder le fix (le testeur peut aider au besoin),
- Le testeur fait une revue de code du développeur,
- Le testeur test le fix, cherche de nouvelles façons de « casser » le fix et vérifie qu'il n'entraine aucune régression (un nouveau bug),
- Lorsque tout est bon le testeur s'assure que le KB – qui va aller sur Windows Update – s'installe/désinstalle correctement et que la DLL contient bien le fix,
- Le testeur crée un test automatisé pour s'assurer que le fix ne sera pas annulé par un autre fix plus tard.

Le testeur s'assure donc de vérifier que le KB que des milliards d'utilisateurs vont recevoir sur Windows Update fonctionne bien et ne casse pas leur machine. Il doit fouilleur dans le code pour comprendre le bug afin de bien comprendre comment le tester. Une partie de mes composants se trouvant en mode kernel, la moindre erreur peut résulter en un écran bleu, il faut donc faire très attention.

La partie de mon travail que je préfère, c'est lorsque nous avons des bugs de sécurité. Le processus est légèrement différent et dans certains cas critiques, l'impact est énorme. Mon travail requiert une très bonne compétence pour le debug (nous utilisons WinDBG) ainsi que de bien comprendre du code juste en le lisant. La partie développement est un peu plus à l'écart, raison pour laquelle j'ai décidé de changer de rôle.

Des conseils aux étudiants et dévs qui nous lisent ?

Suivez la voie qui vous plait le plus et ne vous enfermez pas dans une technologie – comme j'ai pu le faire. Pour ceux qui souhaitent venir travailler dans une entreprise américaine, ne sous-estimez pas les algorithmes et les structures de données, c'est très important. Microsoft se fiche que vous soyez un expert C#/Windows Phone/SharePoint ou autre, ce qui importe c'est votre capacité à réfléchir sur un problème et à le résoudre le plus simplement possible. Le meilleur moment pour postuler est le jour qui suit celui où vous avez reçu votre diplôme; n'attendez pas et surtout ne vous dites pas « Je vais travailler quelques années en France pour avoir de l'expérience », vous allez oublier tous vos cours théoriques, et votre expérience ne comptera pas ici. Pour tous les autres, soyez fier d'être développeur et surtout, devenir « chef de projet » ne doit pas être votre objectif.

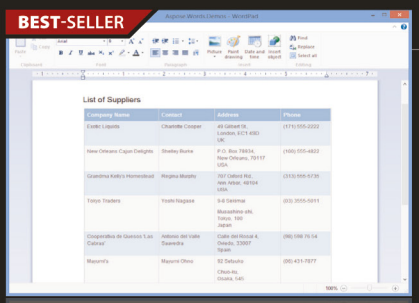
Pour le petit mot de la fin je dirais que je suis toujours aussi fan du métier de développeur et je ne me vois pas changer dans les années à venir (qui a dit « chef de projet » ?). Chez Microsoft les personnes techniques peuvent monter très haut dans la hiérarchie tout en restant très techniques, autrement dit, pas besoin de faire du management. Pour terminer – et malgré toutes les critiques que l'on peut lire – Microsoft est une entreprise très humaine et tout simplement géniale !

Interview réalisée par Julien.

Environnement de travail

Etant le petit dernier dans ma division et manquant d'espace, je partage mon bureau avec un autre testeur. Pour travailler j'ai 3 machines – une machine de développement, une machine de test avec un serveur Hyper-V qui contient plusieurs machines virtuelles de test et une machine de test dans le cas où le bug soit dépendant d'un matériel spécifique. Je travaille avec 2 écrans dont un tactile. Je travaille la plupart du temps avec la porte de mon bureau fermée, mon casque sur les oreilles et de la musique (je n'arrive pas autrement). Mon équipe est plutôt sociable, nous allons manger ensemble tous les midis et au retour, avec certaines personnes, c'est direction la table de baby-foot pour une trentaine de minutes de jeu. Un de mes collègues a une tireuse à bière dans son bureau et certains jeudis ou vendredis soirs, après les heures de travail, on se retrouve dans une salle pour discuter (on discute peu travail) et plaisanter. Les gens avec qui je travaille sont vraiment géniaux !

BEST-SELLER



Aspose.Words for .NET | à partir de € 718

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

BEST-SELLER

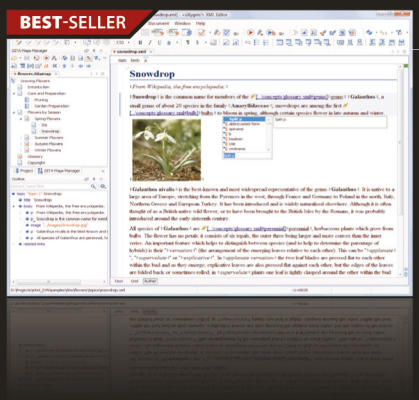
DevExpress DXperience | à partir de € 1 077

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

BEST-SELLER

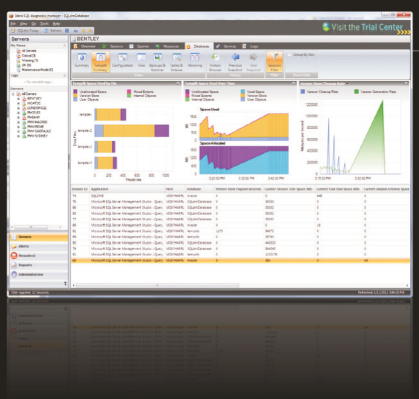


oXygen XML Editor Professional | à partir de € 350

<oXygen/> XML Editor

Éditeur XML multiplateforme supportant la plupart des technologies XML.

- Distribuez/actualisez facilement des plug-in/frameworks pour Oxygen
- Simplifiez la configuration des projets XML avec Master Files
- Affichage des modifications et commentaires de révision dans légendes
- Support d'édition visuelle convivial pour DocBook, DITA, TEI, XHTML
- Validez les documents XML avec les schémas XML, Relax NG, DTD, NVDL et Schematron



SQL Diagnostic Manager | à partir de € 1 719

idera

Suivi des performances SQL 24h/24, 7j/7, alertes et diagnostics.

- Suivi des performances SQL Server physiques et virtuelles
- Analyse approfondie des requêtes pour identifier les délais et les consommations de ressource excessifs
- Planification de la capacité révélant les tendances de croissance des bases de données et réduisant l'utilisation du serveur

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 DISCOVER EUROCARD MasterCard

test de l'imprimante 3D FreeSculpt

Dans Programmez !, nous avions il y a quelques mois évoqué l'impression 3D avec un modèle à monter soi-même. Ce mois-ci, nous avons testé un modèle prêt à l'emploi, la 3D FreeSculpt, distribuée en France par PEARL Diffusion. FreeSculpt est un fabricant allemand.

Premier contact

La bête pèse environ 13 kg et mieux vaut avoir une boîte à lettres adaptée (private joke). Plus sérieusement, l'imprimante s'avère facile à manipuler. Mais il faut tout de même disposer d'un minimum de place pour pouvoir ouvrir la face avant et accéder facilement à la connectique, et surtout au support de la bobine plastique. D'aspect, la 3D FreeSculpt est relativement massive, ressemblant à un gros pavé. Le design est austère mais fonctionnel : un panneau de commande avec un écran. Une connectique très dépouillée : alimentation, ports USB et SD. C'est tout. Deux couvercles transparents permettent d'accéder au plateau et à la tête d'impression. L'imprimante est livrée avec le support et les fixations, les câbles, une carte SD, une spatule (pratique pour décoller les objets imprimés) et 4 ciseaux pour les finitions de l'objet imprimé (il reste toujours des déchets et des imperfections). Les objets ne doivent pas dépasser 225 x 145 x 150 mm.

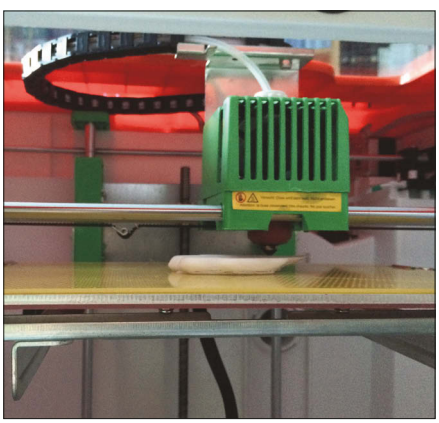
La 3D FreeSculpt ne prétend pas concurrencer les références du marché (notamment les MakerBot) mais pour un tarif attractif, le grand public peut goûter aux joies de l'impression 3D.

Une installation simplissime

Rien de bien compliqué :

- Installation du support (à fixer avec les pinces fournies)
- Coupe des serre-câbles (tête d'impression).

Ensuite, il suffit de brancher le câble d'alimentation et le port USB si besoin. Si la bobine n'est pas installée, il faut l'installer et fixer le fil plastique dans la buse d'impression. Ce qui peut être assez énervant... Une fois tout cela réalisé, il faut lancer un test de calibration (via l'écran et le panneau de contrôle). Ce test permet de vérifier que tout fonctionne.



AVERTISSEMENTS !

- Placer votre imprimante dans un endroit aéré. Les émanations dues à la chauffe du plastique peuvent être gênantes pour les personnes sensibles.
- La qualité du plastique utilisée est cruciale et certains plastiques bas de gamme peuvent être toxiques.
- Ne touchez jamais la buse d'impression ni le filament pendant ou après une impression. Laissez refroidir avant tout nettoyage.
- Ne touchez pas la plaque de support. Le support monte à +60°.
- Respectez les consignes de sécurité et les bonnes pratiques du constructeur.
- Respectez les dimensions maximales d'impression.

Une petite impression !

Pour imprimer, vous pouvez passer uniquement par la carte SD, il suffira alors de copier les fichiers STL. Vous pouvez aussi utiliser Windows ou OS X. Dans ce cas, vous devez installer Pearl 3D, le logiciel de visualisation qui permet de générer des fichiers et du code 3D. Un logiciel de modelage 3D est aussi disponible.

Sur Mac, nous avons rencontré des problèmes avec Pearl 3D, nous avons donc utilisé la version Windows. Pearl 3D permet donc de visualiser les modèles 3D qui doivent être au format STL, format standard pour imprimante 3D.

Le logiciel Pearl 3D a parfois du mal à reconnaître l'imprimante. N'hésitez pas à utiliser un autre câble que celui fourni avec l'imprimante, à vérifier la version des pilotes, et à mettre à jour le firmware. Avant toute impression de Pearl 3D, n'oubliez pas de connecter l'imprimante au logiciel... L'imprimante va alors brièvement se couper et redémarrer. L'icône imprimante va s'activer. Si la buse (et la plaque) n'est pas assez chaude, la 3D Free-Sculpt repassera en mode « chauffage ». Il demande un peu de pratique mais il permettra de réaliser très rapidement des objets. Cela ne remplacera néanmoins jamais une connaissance de la 3D pour réaliser ses propres objets. La génération du modèle peut prendre de longues minutes, tout dépendra de la complexité et des dimensions.

Plusieurs modèles 3D sont disponibles directement depuis la carte SD. Le plus simple est faire un premier test avec un des exemples fournis. Pour le petit lapin, il faut presque 2 heures d'impression. Les petits défauts seront à gratter.

Un gadget geek à 800 € ?

A quoi peut servir une imprimante 3D ? On se rend rapidement compte que cela peut rendre de nombreux services : fabriquer une pièce cassée et difficilement trouvable, créer de nouveaux objets, etc. En réalité, les usages sont très nombreux. Cependant, la qualité d'impression sera primordiale. Ensuite, il faut soit trouver le bon fichier STL, soit le créer. Les catalogues en ligne de modèles 3D se multiplient.

Certains sont payants, d'autres gratuits. Shareways propose une belle variété de modèles mais les tarifs montent parfois très rapidement ! A vous d'imaginer la suite !

François **Tonic**

Les +

- Le prix
- Les pièces détachées
- La simplicité d'usage
- La finition de l'imprimante
- Rapidité du calibrage

Les –

- Les logiciels parfois peu pratiques
- Le support de OS X trop limité
- Une connectique limitée
- Pas de wifi
- Une seule buse d'impression

Une année pleine de technologies et de codes avec PROGRAMMEZ!

le magazine du développeur

www.programmez.com

réservée à la France Métropolitaine

Spécial étudiant 39€

1 an 11 numéros

1 an 11 numéros 49€

seulement (*)

2 ans 22 numéros 79€

seulement (*)

(*) Tarifs France métropolitaine

09-18-12 © DrAfter123 / istock

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

ABONNEMENT retourner avec votre règlement à
Programmez, 17, route des Boulangers 78926 Yvelines cedex 9

☐ ***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 :

Tél : ***(Attention, e-mail indispensable pour les archives sur internet)***

E-mail : @

☐ Je joins mon règlement par chèque à l'ordre de Programmez !

☐ Je souhaite régler à réception de facture

PROG 174

***Offre limitée**, valable jusqu'au 30 mai 2014*

BUILD 2014 : codez et déployez (presque) partout

La conférence développeur BUILD 2014 a permis à Microsoft de repartir à l'offensive sur Windows et Windows Phone, avec les nouvelles versions 8.1. Surtout, l'éditeur a dévoilé de nombreuses nouveautés sur les langages, les frameworks et les outils. Une des plus marquantes est l'application universelle Windows. Le même code pourra fonctionner sur Windows, Windows Phone, et dans une certaine mesure, via Xamarin, sur iOS et Android. Même si cette solution impose des limites et des contraintes (toutes les fonctionnalités et API de Windows Runtime ne sont pas supportées), l'éditeur cherche à se replacer dans la course des apps, et à attirer toujours plus de développeurs.

De nombreuses nouveautés ont aussi été annoncées sur Azure. Nokia a dévoilé les nouveaux Lumia mais durant les deux plénières, les tablettes Surface en furent absentes. Autre élément stratégique : l'open source. Microsoft a décidé d'aller plus loin avec la .Net Foundation. Des dizaines de projets sont d'ores et déjà disponibles. L'objectif est de faciliter l'usage, les

Fig.1

évolutions des projets Microsoft et non Microsoft. La .Net Foundation reprend les principes des fondations Mozilla, Apache. Plus que jamais, Microsoft veut séduire les communautés open source et développer une communauté. Deux projets marquants ont été versées à la fondation : le compilateur Roslyn et WinJS.

Dans ce dossier, revivez les principales annonces, l'ambiance de la BUILD. Nous faisons aussi un focus sur les applications universelles. Les autres nouveautés seront abordées dans les prochains numéros.

Enjoy !

François **Tonic**

Build 2014 : résumé de la conférence développeurs

La 4ème saison de la conférence Microsoft annuelle « Build » pendant laquelle la société annonce toutes ses nouveautés s'est déroulée du 2 au 4 avril au Moscon Center de San Francisco. Fig.1

A nouveau beaucoup de participants pour cette nouvelle édition où des annonces autour du développement des apps sont attendues. Les journées commencent à 7h30 avec un petit déjeuner sur place et direction dès 8h la grande salle où va se dérouler la keynote les 2 premiers jours (pas de keynote le dernier jour). C'est durant ces sessions principales que les grosses annonces sont faites par certaines personnalités de Microsoft.

1ère keynote, les grosses annonces !

La première keynote a mis clairement l'accent sur la nouvelle version de Windows Phone à venir prochainement : Windows Phone 8.1 et les nouvelles fonctionnalités qu'elle apporte Fig.2. On nous présente le nouvel écran de verrouillage, il permet aux applications une personnalisation beaucoup plus importante aussi bien au niveau des informations affichées que de la disposition de ces informations. Les nouveautés de l'écran principal sont ensuite mises en avant avec la possibilité pour l'utilisateur d'afficher plus de tuiles, même sur un téléphone de taille standard, et de personnaliser le fond d'écran via une mosaïque des tuiles présentes. Puis vient l'une des grosses stars de cet évènement : Cortana. L'assistant personnel intelligent du système. Basé sur le moteur bing, l'assistant peut interagir avec l'utilisateur par texte ou par la voix. Il est capable d'apprendre de vos habitudes, de vos contacts et des différents cercles au sein de vos contacts, de vos intérêts, grâce à un accès à vos emails, à vos recherches Web et des lieux que vous avez l'habitude de fréquenter. Tous les accès aux données personnelles sont configurables par l'utilisateur comme l'accès d'ailleurs à d'autres applications ; certaines API permettent en effet à ces dernières d'étendre les fonctionnalités de Cortana. Cet assistant est ainsi capable de vous aider dans certaines tâches courantes comme l'ajout d'un rendez-vous à votre calendrier, ajouter un réveil ou encore effectuer une recherche Web en plusieurs étapes. Une autre grosse nouveauté va également rendre l'utilisation de votre smartphone plus productive : le centre de notifications ou, tel qu'il a été présenté, l'Action Center. Cette nouvelle fonctionnalité permettra d'avoir un accès rapide à différentes actions systèmes comme l'activation/désactivation du Wifi, Blue-


Fig.2

Quelle interopérabilité entre mes différents fournisseurs Cloud ?

Avec Aruba Cloud,

vous avez l'assurance de ne pas être prisonnier d'un fournisseur. Nos services sont intégrés au **driver DeltaCloud** et compatibles **S3**. De plus, vous pouvez utiliser des formats standards d'images de machines virtuelles, **avec VHD et VMDK**, ainsi que des modèles personnalisés provenant éventuellement d'autres sources.

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

Nous avons choisi Aruba Cloud car nous bénéficions d'un haut niveau de performance, à des coûts contrôlés et surtout car ils sont à dimension humaine, comme nous. Xavier Dufour - Directeur R&D - ITMP

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

aruba CLOUD

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

MY COUNTRY. MY CLOUD.*

*MON PAYS, MON CLOUD.

tooth, mode avion, etc. ainsi, bien sûr, qu'aux notifications des applications.

De nouvelles annonces très orientées end user donc, mais l'entreprise n'est pas en reste. Le support du VPN est de la partie avec la configuration des applications qui en tirent parti ainsi que la possibilité de crypter et signer les emails (s/mime). Il est également possible pour les entreprises d'avoir un meilleur contrôle sur les applications et les certificats déployés sur leur flotte mobile.

Nous avons également des nouveautés autour des applications natives. L'application Calendrier est mise à jour avec le support d'une vue semaine plutôt efficace et l'intégration de la météo. L'ergonomie du Store est revue avec des vues plus centrées utilisateur et des suggestions personnelles. Des nouveaux font également leur arrivée dans la famille « Sense » avec une application pour des informations et de la configuration autour de l'utilisation et la gestion de la batterie ainsi qu'une application pour la gestion et le partage du Wifi à certains contacts. A ce sujet, nous avons le droit à une nouvelle application Contact qui apporte notamment la notion de « cercle interne » permettant par exemple de sélectionner un ensemble restreint de vos contacts pour lesquels vous restez joignable même pendant une plage horaire que vous pouvez définir comme silencieuse. Internet Explorer passe en version 11 avec notamment le support de WebGL et d'un mode de lecture.

Windows 8.1 est également mis à l'honneur. L'update de la version 8.1 est disponible pour les abonnés MSDN. Au menu, une meilleure ergonomie de l'utilisation clavier/souris dans l'environnement moderne. Cela se traduit par un accès à la barre des tâches classique depuis une application Windows Store ainsi qu'à une barre de titre permettant de fermer, minimiser ou encore docker l'application. Les actions sont également plus facilement accessibles depuis la souris avec l'ajout d'un menu contextuel semblable à l'expérience du bureau classique remplaçant dans ce contexte l'« Application Bar » Fig.3.

Deux belles mises à jour pour deux systèmes dont les modèles de développement sont proches. Néanmoins certaines parties présentent encore des différences bien distinctes; pour le développeur, cela se traduit par une tâche d'adaptation afin de porter une application d'une plateforme vers l'autre. Microsoft rend cette tâche beaucoup plus aisée avec la 2ème plus grosse annonce de cet évènement : les Universal Apps. Une nouvelle fonctionnalité de Visual Studio 2013 apportée par l'update 2 permettant d'avoir accès entre-autres à de nouveaux templates de projets pour le développement conjoint d'applications Windows 8 et Windows Phone 8.1. Cela se traduit dans un premier temps par un nouvel élément dans l'explorateur de solution : un nœud « shared ». Son but est de rassembler tous les éléments à partager : le code déclaratif et impératif ainsi que différentes ressources et assets. Le taux de code partagé est beaucoup plus important grâce à la convergence de l'api WinRT maintenant disponible avec Windows Phone 8.1. Même le designer joue le jeu avec une nouvelle interface hybride où le développeur peut travailler sur une même vue dans les 2 contextes.

Cette première keynote se termine avec une visualisation intéressante des projets en cours de Microsoft. On voit une démo d'Office en mode « Moderne », une solution Universal App Xaml qui cible Xbox One et des annonces sur Kinect 2 pour desktop, DirectX 12 et l'Internet of Things mais nous allons y revenir ;) Fig.4

Lors de la 2ème keynote et après une première journée consacrée aux expériences clientes et à la convergence entre téléphone, tablette et télévision, Microsoft a tout naturellement axé la communication sur le Cloud, afin de compléter ses annonces autour de la vision «Mobile First, Cloud First». Pour compléter les annonces autour d'Azure, Microsoft a aussi voulu remettre les pendules à l'heure en ce qui concerne .Net : alors que beaucoup prophétisaient la mort de .Net à plus ou moins long terme, Microsoft sort l'artillerie lourde avec des nouveautés qui viendront certainement bouleverser la vie des développeurs C# dans les mois et années à venir : Ryu JIT, .Net Native et Roslyn Fig.5.

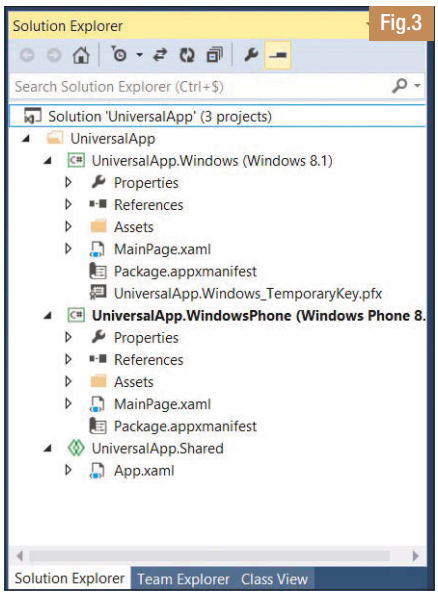


Fig.3

Internet of Things

Un sujet particulièrement *trendy* ;) Le développement autour de « l'Internet des objets ». On a le droit à plusieurs démonstrations claires, simples et efficaces d'applications Windows Phone interagissant avec des objets de différentes tailles (un feu tricolore par exemple). Un focus est fait sur le .Net Micro Framework utilisé pour ces différentes démonstrations. Il s'agit d'un framework open source maintenu par Microsoft permettant de développer et de déployer des applications sur des périphériques aux ressources limitées. Cela permet par exemple de fournir un accès à ce type de périphérique à des applications mobiles grâce à un serveur sur ce périphérique développé avec le .Net Micro Framework. L'accent est mis sur l'investissement constant fait par Microsoft sur ce framework pour l'améliorer notamment au niveau des performances.

L'autre point très intéressant présenté est la dernière version de Windows Embedded. Windows Embedded est une version de Windows adaptée au milieu de l'industrie et des périphériques embarqués. Dans sa dernière version, on distingue Windows Embedded 8.1 pour Windows 8.1 et Windows Embedded Handheld pour Windows Phone. Chacun de ces systèmes proposent des api qui correspondent aux standards de l'industrie, ainsi que des fonctionnalités spécifiques pour l'optimisation de l'utilisation en entreprise ou d'un contexte embarqué. On pourra par exemple restreindre les applications et les paramètres disponibles selon le rôle de l'utilisateur sur le périphérique, on va pouvoir désactiver certaines fonctionnalités systèmes comme la « charm » bar ou encore interdire la connexion d'un périphérique USB.

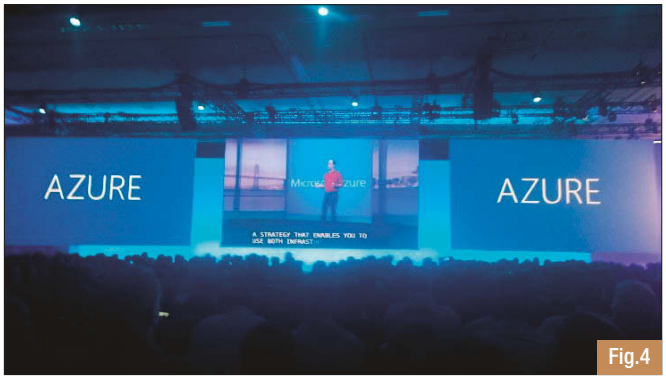


Fig.4

A noter également un point intéressant, la gestion du Bluetooth LE pour Windows Phone 8.1 (pas encore disponible pour la developper preview mais la fonctionnalité sera bien présente lors de la RTM) avec notamment de nouveaux triggers pour la gestion de la déconnexion Fig.6.

On a droit à une vue d'un concept de Windows Embedded 8 embarqué

dans une voiture (un potentiel successeur de Windows Embedded Automative ?). On peut voir un système embarqué avec une interface « moderne » mais conçu pour une utilisation adaptée à la voiture, i.e. des interactions externes mises en avant comme des implémentations poussées de la voix ou de commandes externes.

Qualité et performances

Comme indiqué durant la première keynote, avec le nouveau tooling apporté par l'update 2 de Visual Studio 2013 pour le développement des apps, Microsoft poursuit sa volonté de fournir aux développeurs l'environnement de développement le plus performant possible. Les performances sont une priorité forte, et encore plus dans le domaine du développement mobile. Dans ce sens, plusieurs sessions de cet évènement étaient consacrées à ces sujets.

Pas mal de changements côté performances sur Windows Phone 8.1, de gros efforts ont été faits sur le temps de chargement, la consommation mémoire, le scrolling, la virtualisation et la latence du touch. Plusieurs démonstrations ont été faites comparant une application Windows Phone 8.0 avec une application Windows Phone 8.1, sur une même vue par exemple, l'application passait de 50 mo de consommation mémoire sous 8.0 à environ 15 mo avec le nouveau modèle XAML + WinRT de la 8.1 !

La partie testing n'est pas en reste, au contraire. Tout le moteur de tests unitaires a été porté pour les applications XAML, i.e un nouveau template Visual Studio permet maintenant de créer un projet de test correctement contextualisé et intégré. Un test unitaire Windows Phone par exemple pourra lancer l'émulateur, et le résultat du test sera visible dans le « Test Explorer »; ceci permet de tester des cas particuliers (pas de réseau, changement de cultures, etc.). Les Coded UI Tests sont également de la partie avec des templates de projets dédiés. Une API permet de simuler les gestures ou les boutons physiques afin de reproduire des scénarios utilisateur. Un outil UI nous aide même dans cette tâche en générant une partie du code nécessaire au test UI à partir des éléments que l'on sélectionne à l'écran. A noter que la fonctionnalité de « step recorder » permettant d'enregistrer un scénario complet n'est pas disponible pour les Windows/Windows Phone Store apps.

Azure en fanfare !

Lors de la seconde keynote, Scott Guthrie a commencé son intervention avec 2 retours d'expériences sur l'année passée pour Microsoft Azure. 2 exemples particulièrement bien choisis car traitant de types de Workloads très difficilement gérables sans la flexibilité du Cloud.

Le premier est Titanfall : Il s'agit d'un jeu vidéo massivement multi-joueurs lancé sur Xbox One, Xbox 360 et PC où l'IA des personnages non-joueurs ainsi que l'hébergement des parties sont entièrement gérés par Azure, permettant ainsi de libérer de la puissance sur les machines clientes pour privilégier le moteur de rendu et la fluidité générale du jeu. Cela permet aussi d'avoir une IA beaucoup plus intelligente, et plus facilement évolutive, car elle est déportée sur des machines dédiées, faciles à mettre à jour. Une des grandes démonstrations de la pertinence du Cloud sur ce jeu, a aussi été la manière dont la charge du jour de lancement a été absorbée sans accrocs (à une époque où quasiment tous les gros jeux multi-joueurs connaissent des problèmes techniques à la sortie). Il faut dire que Microsoft n'a pas lésiné sur les moyens : pour la journée de lancement, 100000 VM Azure étaient prêtes à accueillir les joueurs ! Et bien sûr, après la fièvre du premier jour, les développeurs ont pu diminuer le nombre de serveurs pour réduire leurs coûts. Une belle manière de montrer comment Azure peut répondre à une très forte charge, avec des pics d'activité exceptionnels, tout en évitant le sur-provisionnement de serveurs.

Un 2ème exemple là aussi assez impressionnant en termes de chiffres, a été la couverture des JO de Sotchi avec l'encoding de toutes les épreuves ainsi que leur diffusion en live et en VOD par la plateforme Azure Media Services pour le compte de NBC. En termes de charge, on parle tout de même là de plus 2 millions d'internautes connectés en simultané simplement sur la diffusion live de la demi-finale de hockey sur glace USA-Canada ! Et dans le même temps, il fallait bien sûr que les quelques utilisateurs intéressés par les épreuves de Curling ne soient pas perturbés. Si on cumule ces besoins de scalabilité avec la nature «évènementielle» et très limitée dans le temps des JO, on imagine assez mal comment l'on aurait pu monter une infrastructure capable de relever le défi en gardant des coûts raisonnables sans la flexibilité du Cloud !

Azure, on simplifie tout !

Tout d'abord, un des premiers messages importants est la simplification de la mise en place des services, que ce soit en termes d'API ou en termes d'expérience pour le développeur. En terme d'API, les Cloud Services (types de services où le développeur à le contrôle le plus fin, mais qui demande souvent des adaptations au niveau du code) semblent un peu à l'abandon au profit d'Azure Web Sites, Azure Mobile Services, et des WebJobs. Ainsi Azure WebSites continue à évoluer, avec l'autoscaling en « general availability », le support de Traffic Manager, ou encore de Java.

Du côté du tooling, on remarque que de plus en plus de choses sont disponibles directement depuis Visual Studio : templates de projets Web liés à un WebSite Azure, environnement de simulation pour Mobile Services .Net, création de VMs directement depuis VS etc.

Azure, le Cloud parfait pour les solutions d'entreprise

Un des gros obstacles à la migration vers des solutions Cloud publiques pour une entreprise (ou pour un éditeur de solutions B2B), est la gestion de l'annuaire des utilisateurs. Microsoft profite de la Build pour annoncer la « general availability » d'Azure AD Premium. En plus de cela, Azure AD offre maintenant la possibilité de donner le droit à des applications tierces d'accéder aux API Office 365. Les scénarios ouverts sont très intéressants, avec la possibilité de faire du Single Sign-on entre les applications

Fig.5

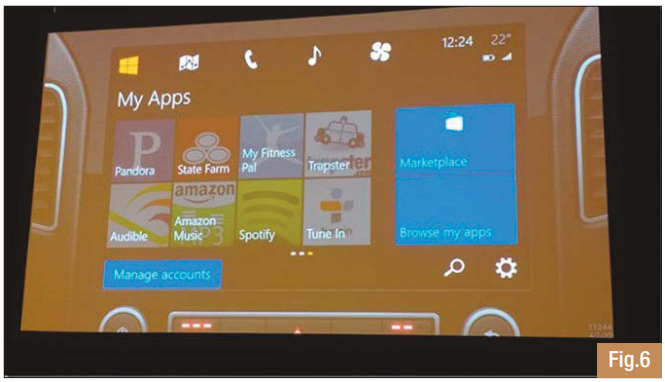


Fig.6

Office et les applications d'entreprises, et de faire interagir tout ce petit monde. Ainsi une application métier pourra poster un document dans une liste Sharepoint en utilisant l'identité de l'utilisateur connecté, envoyer un mail, générer un notebook OneNote, etc.

Mobile First, Cloud First, où se situe Azure dans tout ça ?

Dans la stratégie « Mobile First, Cloud First » de Microsoft, Azure a une place prépondérante. En effet, Microsoft présente sa solution Cloud comme étant la plateforme d'hébergement la plus adaptée aux APIs avec lesquelles communiquent les applications mobiles. Ainsi, en plus de NodeJS, Azure Mobile Services supporte désormais .Net avec ASP.Net Web API et des packages Nuget pour implémenter les TableController de la même manière qu'avec NodeJS. Au niveau outillage, Visual Studio 2013 update 2 intègre un environnement de simulation de Mobile Services, permettant de debugger en local, et il est désormais également possible de débugger des services mobiles tournant dans Azure à distance, depuis Visual Studio ! Autre nouveauté, un connecteur Windows Azure Active Directory est désormais disponible, pratique pour développer des applications mobiles d'entreprise ou des applications B2B (sous Windows, IOS ou Android) ! Du côté des sdk client pour Mobile Services, une preview supportant le mode déconnecté est désormais disponible.

De son côté, Notification Hub continue à supporter toujours plus de plateformes, et les devices Amazon Kindle font désormais partie du lot.

.Net is back !

En marge des annonces autour d'Azure, Microsoft a voulu aussi rassurer les développeurs .Net : oui, .Net reste la plateforme de développement de prédilection chez Microsoft, et oui, Microsoft continue d'investir massivement dans cette plateforme.

Peut-être sous l'influence de Satya Nadella, un vent de transparence semble souffler très fort sur les équipes du Framework. Ainsi, l'équipe CLR dévoile ses plans à moyen/long terme avec des previews sur RyuJIT et .Net Native très en avance de phase.

La documentation MSDN intègre des liens vers le code source du Framework, et Microsoft lance en partenariat avec Xamarin et différents acteurs de la communauté Open Source .Net la .Net Foundation, dont le but est de promouvoir et d'aider le développement de projets Open Source de haute qualité, provenant de Microsoft (Roslyn, EF, ASP.Net,...) ou de la communauté.

Le futur de C# : Roslyn

Roslyn, on n'en a parlé déjà en long et en travers, est le nouveau compilateur pour C# et VB.Net. Sa grande force étant de se comporter non pas comme une boîte noire, mais comme un service permettant d'obtenir des informations sur la sémantique du code.

Roslyn n'est malheureusement pas encore terminé (mais c'est en bonne voie visiblement), et son développement vient d'être passé en Open Source. En plus d'augmenter la transparence (on peut voir passer les commits des développeurs), cela ouvrira des scénarios sympathiques : on pourra non seulement proposer des correctifs ou des fonctionnalités à Microsoft, mais on pourra surtout modifier le langage suivant ses besoins, et l'adapter à un domaine particulier ! Le projet Roslyn commence déjà à être forké en masse, on peut donc s'attendre à voir apparaitre des déclinaisons de C# / VB.Net originales...

Ryu JIT, un compilateur JIT qui met des grosses baffes

Depuis sa première version qui ne ciblait que les processeurs X86 sous Windows, .Net a progressé pour devenir un véritable Framework multiplateforme (Windows Desktop, Windows Store, Windows Embedded, Windows Phone, Silverlight, Xbox 360, Xbox One, le tout sous X86, X86-64, différentes version d'ARM, de PowerPC...). Microsoft se retrouve donc maintenant dans une situation où il doit maintenir un nombre de JIT compilés spécifiques à chaque plateforme / architecture processeur important. Cela ralentit l'évolution de la CLR, et Microsoft a donc décidé de revoir en profondeur ses JIT compilers et de démarrer le projet RyuJIT.

RyuJIT est donc un nouveau JIT compiler complètement modulaire, avec un focus sur la mise en commun d'un maximum de code quelle que soit la plateforme ciblée. Le but étant de pouvoir introduire des nouveautés à la CLR sans avoir à modifier 15 implémentations différentes du JIT compiler.

Pour démontrer la promesse d'une plus grande agilité de la CLR, la preview actuelle de RyuJIT intègre une fonctionnalité inédite : le support des instructions SIMD (single instruction, multiple Data).

Ces instructions sont très utiles pour les scénarios propices au « Data Parallelism », et sont notamment très utilisées dans le jeu vidéo, ou dans les applications scientifiques.

Sur des processeurs modernes les gains théoriques par rapport au jeu d'instruction classique pour des traitements numériques vont de x4 (pour les processeurs supportant SSE2 et ARM NEON) à x8 (pour les processeurs supportant AVX).

.Net Native

En parallèle à Ryu JIT, Microsoft propose aussi un nouveau mode de compilation pour les applications .Net Windows Store et Windows Phone : la compilation native.

De prime abord, on pourrait imaginer qu'il ne s'agit que d'un renommage de NGEN, mais ce n'est pas du tout le cas : là ou NGEN ne fait que pré-jitter du code .Net (en utilisant le même compilateur que le compilateur JIT), .Net Native va prendre l'intégralité du code MSIL ainsi que de son graphe de dépendance, et l'exposer sous un format que le linker et optimiseur C++ est capable de reconnaître. C'est alors le compilateur C++ et non plus le JIT .Net qui va s'occuper d'optimiser et de générer le code assembleur. Le code produit est un exécutable contenant une CLR indépendante capable d'exécuter le code natif obtenu, self contained : aucune dll .Net (y compris provenant de la BCL) ne sera chargée à l'exécution, car tout le code .Net utilisé par l'application aura été statiquement linké à l'exécutable.

Là où c'est intéressant, c'est que seul le code réellement utilisé sera linké : cela diminue le temps de chargement et la consommation mémoire de façon très significative. De la même manière, comme tout le code .Net, qu'il provienne de l'application elle-même ou du Framework, est linké dans le même exécutable; les opportunités d'optimisation sont très grandes. Par exemple, on va pouvoir inliner du code provenant du Framework directement dans le code de l'application ! Un autre aspect intéressant est que là où le compilateur JIT est assez récent et est optimisé pour la génération rapide de code à l'exécution, le compilateur C++ bénéficie d'une 30aine d'années d'expérience et est optimisé pour générer le code le plus rapide possible à l'exécution (quitte à allonger de manière très considérable le temps de compilation). Il introduit donc pas mal d'optimisations impossibles à réaliser par le compilateur JIT : auto-vectorization et auto-parallélisation de boucles, ré ordonnancement d'instructions, etc.

En résumé, certainement un des évènements Microsoft avec le plus d'annonces. Ceci que ce soit concernant le développement client, où les efforts faits notamment au niveau de la convergence tendent à montrer un 1er niveau de maturité de ses modèles de développement, qu' au niveau serveur, avec une simplification accrue des implémentations et des outils plus pertinents et productifs pour des réponses plus efficaces à de vrais problèmes. Bref on était contents d'y être et on attend la suite avec impatience ;).

Simon & Arnaud
Infinite Square

À la découverte des Universal Windows Apps

C'est lors de sa grand-messe annuelle dédiée aux développeurs, la Build, que Microsoft a annoncé les Universal Windows Apps, un nouveau type de projet permettant de développer une application unique à destination de Windows, Windows Phone et Xbox. La convergence des plateformes s'accélère chez l'éditeur. Découvrez dans cet article comment tirer parti de cette nouvelle opportunité.

Une application, trois plateformes

Ce n'est plus un secret pour personne, Microsoft souhaite faire converger ses plateformes afin de fournir à ses utilisateurs une expérience unifiée sur tous leurs devices. C'est également l'objectif de l'équipe Visual Studio : fournir un environnement de développement et une plateforme unique permettant de développer une application pour plusieurs usages. Si cet objectif n'est pas encore atteint, Microsoft s'en rapproche toutefois rapidement. C'est début avril, lors de la Build 2014, que le géant des logiciels a annoncé l'arrivée des Universal Windows Apps.

Sur le papier, l'objectif des Universal Windows Apps (UWA) est bel et bien de proposer aux développeurs de créer une application unique qui ciblera différentes plateformes : Windows, Windows Phone et Xbox. Sur le papier seulement, car nous n'en sommes pas encore là. En effet, l'Update 2 de Visual Studio 2013, dont vous aurez besoin pour développer des UWA, ne permet pas pour l'instant de développer pour Xbox. Quant aux deux autres plateformes, Windows et Windows Phone, même si Microsoft annonce qu'il est maintenant possible de factoriser près de 90 % du code de votre application, il sera toujours nécessaire de créer des vues qui s'adaptent à la plateforme de destination.

Le modèle de programmation des UWA est le même que celui des applications Windows Store, il est basé sur le Windows Runtime. Il est ainsi possible d'écrire une application UWA avec des langages managés, comme C# ou VB.NET, des langages interprétés comme JavaScript, ou des langages natifs comme le C++ (Fig.1). Les développeurs Windows 8 ne seront donc pas perdus, puisque les API qu'ils connaissent déjà sont les mêmes que celles utilisées pour développer des UWA. Pour les développeurs Windows Phone, en revanche, il y a plus de changement. En effet, les applications Windows Phone 8.1 se basent maintenant entièrement sur le Windows Runtime, contrôles compris. Attention toutefois, Windows et Windows Phone n'utilisent pas exactement la même version du Windows Runtime. Certains contrôles qui existent sur Windows n'existent pas sur Windows Phone et inversement.

A la découverte des Universal Windows Apps

Entrons dans le vif du sujet. Une Universal Windows Apps n'est ni plus ni moins qu'une solution composée de trois projets : une application Windows 8.1, une application Windows Phone 8.1 et un projet partagé (Shared) (Fig.2). C'est d'ailleurs ce projet partagé qui fait toute la force des Universal Windows Apps. Il s'agit d'un nouveau type de projet permettant de partager facilement des sources entre vos applications Windows et Windows Phone. Le projet partagé ne se compile pas ; il s'agit simplement d'un conteneur qui regroupe tous vos fichiers de code communs aux différentes plateformes.

A la compilation, tous les fichiers contenus dans le projet partagés sont copiés et compilés sur chaque plateforme source.

Le projet partagé accepte tous types de fichier : vos sources C#, JavaScript, C++, vos vues XAML, vos images, vos ressources (resw) ou encore vos fichiers de données (.xml, .json).

Concrètement, lorsque vous déposez un fichier dans ce projet partagé, ce fichier se retrouve à la fois utilisable depuis vos applications Windows 8.1 et Windows Phone 8.1. A l'inverse, depuis le projet partagé, vous pouvez accéder depuis le code à tous les éléments de vos deux applications. Cela peut paraître déroutant au début, mais vous allez voir que Microsoft a encore une fois tout fait pour nous faciliter le travail.

Visual Studio propose différents templates de projets pour débuter avec les Universal Windows Apps :

- Un template blanc constitué d'une application Windows 8.1 vide, d'une application Windows Phone 8.1 vide et d'un projet partagé,
- Un template Hub constitué de ces trois projets avec une page principale reprenant le principe du hub,
- Un template pour créer des librairies,
- Un template pour créer des composants WinRT.

Tout comme les applications Windows 8.1, les applications Windows Phone 8.1 utilisent désormais les contrôles XAML situés dans le namespace Windows.UI.Xaml. Les contrôles utilisables pour développer nos vues sont donc exactement les mêmes sur les deux plateformes.

C'est ce qui va nous permettre de créer des fichiers XAML communs entre nos différentes applications. Windows Phone 8.1 adopte également le même cycle de vie que les applications Windows 8. Ainsi, lorsque vous créez une application Universal Windows Phone en utilisant un template blanc, vous constaterez que le fichier App.xaml est directement partagé. La structure des pages est également la même. Ainsi, il est possible d'écrire le code suivant dans un fichier XAML et de le placer dans le projet partagé afin de le rendre utilisable depuis les deux plateformes.

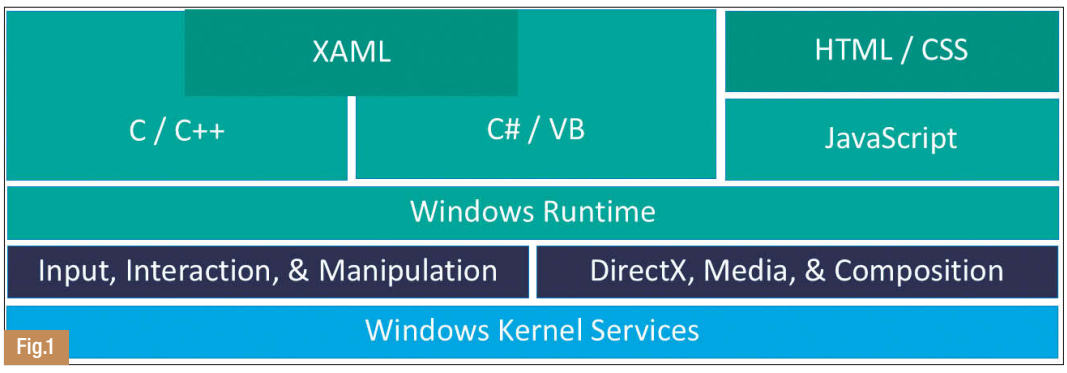


Fig.1

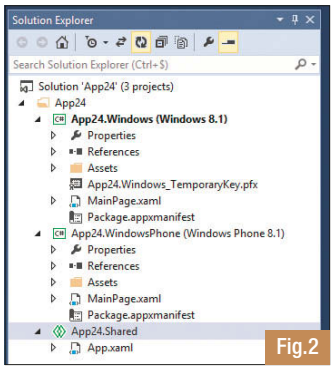


Fig.2

```
<Page
  x:Class="App24.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:App24"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility
/2006"
  mc:Ignorable="d">

  <Grid Background="{ThemeResource ApplicationPageBackground
ThemeBrush}">
    <Button Content="Hello World !" />
  </Grid>
</Page>
```

Ce code est très simple, il affiche simplement un bouton. Qu'en est-il des pages plus complexes ? En théorie, la plupart des pages pourront être partagées étant donné que les contrôles utilisés sont les mêmes. En pratique, on se rendra rapidement compte que ce n'est pas toujours le cas.

Avec les Universal Windows Apps, on peut distinguer trois types de contrôles XAML :

- Les contrôles basique, style Button, CheckBox ou Slider qui s'afficheront exactement de la même façon sur les deux plateformes,
- Les contrôles optimisés, comme le DatePicker ou la CommandBar (anciennement AppBar sur Windows Phone) donc le code sera commun, mais dont la représentation graphique diffèrera en fonction de la plateforme cible,
- Les contrôles spécifiques qui n'existent que sur une seule plateforme. C'est notamment le cas de l'AutoSuggestBox, un nouveau contrôle Windows Phone 8.1, qui n'a aucun équivalent côté Windows 8.1.

Par ailleurs, certains contrôles Windows Phone sont désormais remplacés par des contrôles WinRT. Ainsi le Hub remplace le Panorama, le SemanticZoom remplace le LongListSelector, la WebView remplace le WebBrowser et le SwapChainPanel remplace DrawingSurface et DrawingSurface BackgroundGrid.

Les techniques de partage

Si le projet partagé facilite considérablement la mutualisation de code entre Windows et Windows Phone, il est toujours nécessaire d'appliquer plusieurs techniques pour maximiser le partage.

La première d'entre elle est l'utilisation des directives de compilation conditionnelle. Pour rappel, ces dernières permettent d'utiliser au sein du code des directives permettant au compilateur de prendre en compte ou non certaines portions du code lors de la compilation. Ainsi, il est possible d'écrire au sein d'un même fichier du code qui ne sera compilé que pour Windows, et du code qui ne sera compilé que pour Windows Phone. L'arrivée des Universal Windows Apps fait apparaître deux nouvelles directives : WINDOWS_APP et WINDOWS_PHONE_APP. Ainsi, le code suivant situé dans le code behind d'une vue du dossier partagé permet d'afficher les charmes sur Windows 8 et de naviguer vers la page Page1 sur Windows Phone.

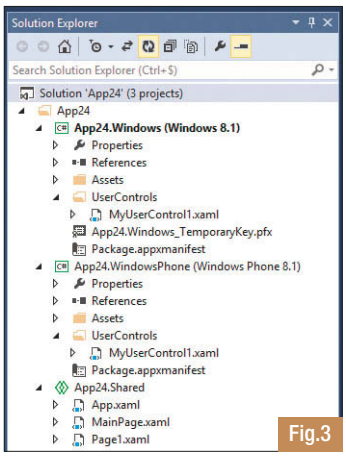


Fig.3

```
private void Button_Click(object sender, RoutedEventArgs e)
{
  #if WINDOWS_APP
    Windows.UI.ApplicationSettings.SettingsPane.Show();
  #else
    this.Frame.Navigate(typeof(Page1), null, new Windows.UI.
Xaml.Media.Animation.SlideNavigationTransitionInfo());
  #endif
}
```

Dans l'exemple ci-dessus, l'utilisation des directives de compilation est indispensable pour deux raisons. Premièrement, car une action différente est réalisée sur chaque plateforme, il est donc impossible d'écrire un code commun. Deuxièmement, car l'API SettingsPane n'existe pas sur Windows Phone ; l'application ne compilerait donc tout simplement pas sans compilation conditionnelle.

Si vous souhaitez créer des vues différentes pour chaque plateforme tout en ayant une page commune située dans le projet partagé, vous allez devoir définir des user control spécifiques sur chaque plateforme. Cela se déroule en trois étapes :

- Créer un dossier situé au même niveau de l'arborescence des applications Windows et Windows Phone,
- Créer dans ce dossier un User Control portant le même nom dans les deux applications,
- Dans la vue du projet partagé, déclarer le namespace commun des User Control et utiliser le contrôle en question.

Notez que lors de la création de fichiers dans une Universal Windows Apps, les namespaces utilisés sont communs. Ainsi, si vous créez une classe Class1 dans le dossier Model du projet MonApp.Windows, son namespace ne sera par MonApp.Windows.Model, mais MonApp.Model. Ainsi, lorsque vous créez deux fichiers identiques dans vos projets Windows et Windows Phone, ces derniers partagent le même namespace. C'est ce qui permet d'y faire référence au sein du projet partagé. (Fig.3)

L'exemple suivant montre deux user control spécifiques (le XAML a été volontairement simplifié) utilisés au sein d'un projet partagé.

Côté Windows :

```
<UserControl
  x:Class="App24.UserControls.MyUserControl1" >

  <Grid Background="Red">

  </Grid>
</UserControl>
```

Coté Windows Phone :

```
<UserControl
  x:Class="App24.UserControls.MyUserControl1" >

  <Grid Background="Green">

  </Grid>
</UserControl>
```

Projet partagé :

```
<Page
  x:Class="App24.MainPage"
  xmlns:uc="using:App24.UserControls">

  <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
```

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 50
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 50
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

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

```
    <uc:MyUserControl1 />
  </Grid>
</Page>
```

Ainsi, en exécutant cette application, on se retrouvera avec une grille rouge sur Windows et une grille bleue sur Windows Phone. Le rendu est différent, mais le code est bel et bien factorisé. En pratique, il sera souvent nécessaire d'avoir recours à cette technique de partage si vous souhaitez créer des interfaces s'adaptant parfaitement à chacune des plateformes cibles. Il est également possible d'accéder à des librairies tierces ou des packages NuGet depuis le projet partagé. Pour cela, il est nécessaire d'ajouter les références aux librairies en question depuis l'ensemble des applications cibles. Par exemple, pour utiliser json.net depuis le projet partagé, il est nécessaire d'ajouter le package NuGet Newtonsoft.Json dans le projet Windows et dans le projet Windows Phone. Cela implique donc que la librairie soit identique sur les deux plateformes.

L'intégration Visual Studio

L'équipe de Visual Studio n'a pas fait les choses à moitié et nous propose avec l'Update 2 de Visual Studio 13 un ensemble d'outils qui va faciliter notre nouvelle vie de développeurs UWA.

Vous le savez désormais, la version WinRT ciblée par Windows et Windows Phone est sensiblement différente. Aussi, il n'est pas impossible que vous utilisiez de temps en temps des API spécifiques au sein du projet partagé. Lorsque c'est le cas, IntelliSense affichera un petit warning jaune à côté des APIs en question afin de vous indiquer que l'API que vous souhaitez utiliser n'est pas commune aux deux plateformes (Fig.4). C'est typiquement dans ce genre de cas que vous devrez avoir recours aux directives de compilation conditionnelle.

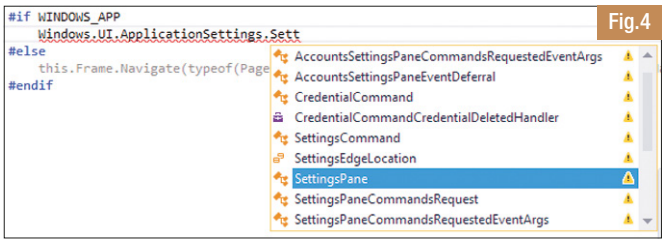


Fig.4

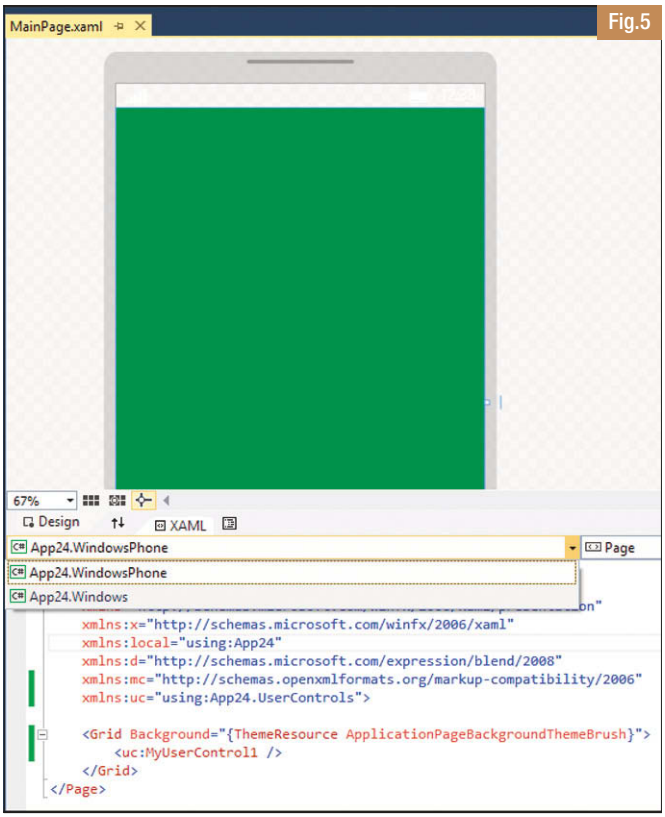


Fig.5

Autre nouveauté, une barre de navigation permettant de passer rapidement d'un contexte Windows à un contexte Windows Phone au sein du projet partagé. Côté XAML, cette barre de navigation permet d'afficher la représentation graphique de la page rapidement sur Windows et Windows Phone (Fig.5). Côté code behind, ce changement de contexte agit directement sur IntelliSense. Ainsi, si vous sélectionnez Windows, IntelliSense vous proposera des API spécifiques à Windows et ignorera les spécificités Windows Phone. Ce changement de contexte permet également de mettre en évidence les sections soumises à la compilation conditionnelle : le code concerné par la plateforme choisie apparaîtra en couleur, tandis que le code non compilé sera grisé.

Il est également possible de choisir le projet de démarrage directement depuis le menu de debug (Fig.6). Ce nouveau menu permet de passer rapidement d'un projet à un autre, d'un simulateur/émulateur à un autre, sans devoir passer obligatoirement par l'explorateur de solution comme c'était le cas auparavant.

Migrer une application existante vers une UWA

Développer une Universal Windows App n'implique pas de reprendre à zéro tous vos projets existants. En effet, vous allez pouvoir migrer – plus ou moins facilement - vos applications vers des UWA.

Côté Windows, vous devrez posséder une application Windows 8.1 pour effectuer une migration. Si vous possédez une application Windows 8, vous devrez préalablement la migrer vers 8.1 avant de la transformer en Universal Windows Apps. Pour effectuer la migration, effectuez un clic droit sur votre projet Windows 8.1, puis sélectionnez « Add Windows Phone 8.1 » (Fig.7). En effectuant cette action, Visual Studio ajoutera automatiquement à votre solution un projet Windows Phone 8.1 et un projet partagé. Le projet partagé créé sera vide. En effet, Visual Studio n'essaye pas de savoir quel partie de votre code peut-être commune ; à vous de vous en occuper. De manière générale, vous allez pouvoir déposer dans le projet partagé tous vos assets, ressources, models et helpers. Difficile toutefois de définir des guidelines, tout projet étant différent.

Si vous n'avez pas suivi la Build, sachez qu'il existe maintenant deux types d'applications Windows Phone 8.1 :

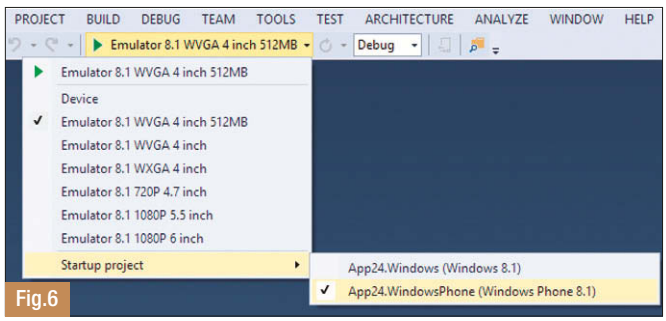


Fig.6

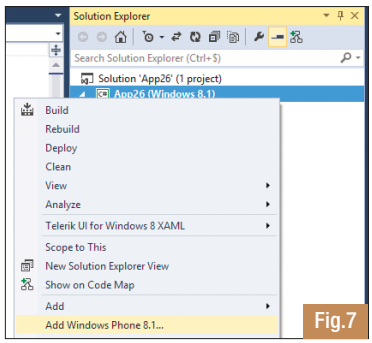


Fig.7

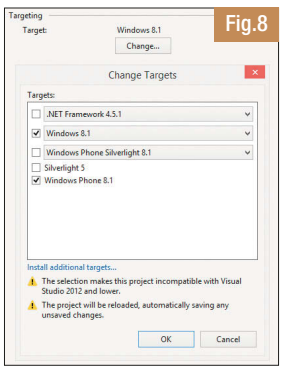


Fig.8

- les applications Windows Phone Silverlight qui sont dans la continuité des applications Windows Phone 8. Celles-ci continuent de s'appuyer sur Silverlight pour le XAML.
- Les applications Windows Phone Store Apps qui utilisent le Windows Runtime.

C'est ce dernier type d'application que vous devrez posséder pour effectuer une transformation de votre projet en UWA. Si vous possédez une application Windows Phone 8 ou Windows Phone Silverlight 8.1, vous serez obligé de créer un nouveau projet from scratch. Si vous possédez une application Windows Phone Store App 8.1, en revanche, vous aurez une option permettant d'ajouter un projet Windows 8.1 à votre solution. En réalisant cette action, Visual studio ajoutera automatiquement un projet Windows 8.1 et un projet partagé à votre solution. Vous aurez alors à nouveau la liberté de choisir quels fichiers partager entre vos applications.

Projet partagé, PCL et librairies

Si vous êtes un adepte du partage de code entre différentes plateformes, vous connaissez probablement les Portable Class Librairies (PCL). Avec la dernière update de Visual Studio 2013, les PCL évoluent et peuvent dorénavant utiliser les API WinRT lorsque les plateformes cibles des PCL sont Windows 8.1 et Windows Phone 8.1. Les PCL peuvent donc désormais inclure du code et des fichiers XAML.

Mais alors, quelle est la différence entre un projet partagé et une PCL ?

La différence est que le projet partagé n'est pas un assembly à part entière, mais simplement un conteneur. Ainsi, le code qu'il contient est compilé séparément sur chaque plateforme cible.

Les PCL, en revanche, sont de vraies librairies. Le code qu'elles contiennent est compilé une seule fois pour l'ensemble des plateformes cibles. Ainsi, il est impossible d'utiliser des directives de compilation conditionnelle au sein des PCL. De plus, les PCL ne permettent pour l'instant que de partager du code managé (C# et VB.NET), tandis que le projet managé peut inclure du code C++ et JavaScript.

La dernière mise à jour de Visual Studio permet également de transformer facilement des librairies en PCL. Depuis les propriétés d'un projet de type librairie, vous trouverez désormais une section « Targeting ». Celle-ci contient un bouton « Change » permettant à la librairie de devenir portable en choisissant une liste de plateformes cibles (Fig.8). Attention toutefois, le code de la librairie n'est pas compilé avant la transformation. Cela signifie que si votre librairie contient du code spécifique à une plateforme avant la migration, ce code ne compilera plus une fois le projet transformé en PCL.

Pour résumer :

Le projet partagé est créé automatiquement lors de la création d'une UWA. Il est indispensable et permet de partager tout type de fichiers,

Les PCL peuvent être utilisées uniquement pour partager du code commun à toutes les plateformes auxquelles la librairie est destinée,

Les librairies ne changent pas, mais peuvent désormais être facilement transformées en PCL.

Conclusion

Les Universal Windows Apps représentent un pas de plus pour Microsoft dans la convergence des plateformes. Si l'on est encore loin de l'objectif final d'écrire un seul code pour toutes nos applications, on notera toutefois que la firme de Redmond avance rapidement et plutôt dans le bon sens.

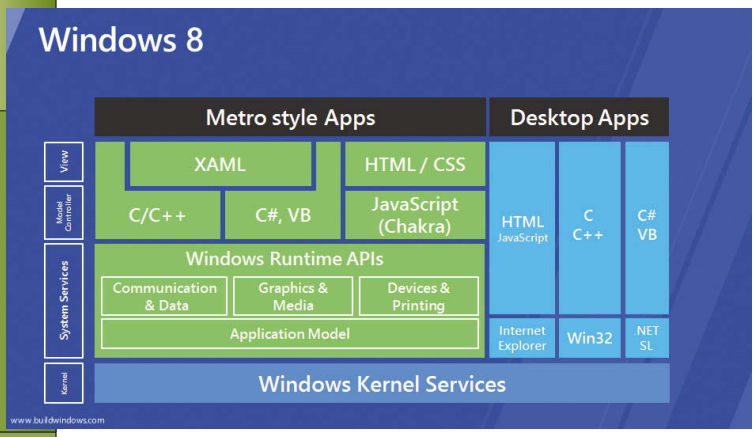
Lors de la Build, Microsoft a d'ailleurs annoncé que cette vision s'accompagnerait également à terme d'un Store unique. Ce but se concrétise par la possibilité pour les développeurs de lier leurs applications Windows et Windows Phone de façon à ce que les utilisateurs n'aient qu'un achat à réaliser pour toutes leurs plateformes. Ainsi, l'achat d'une application Windows 8 permet également d'obtenir la version Windows Phone. Cela fonctionne également avec les achats au sein de l'application (in app purchases).

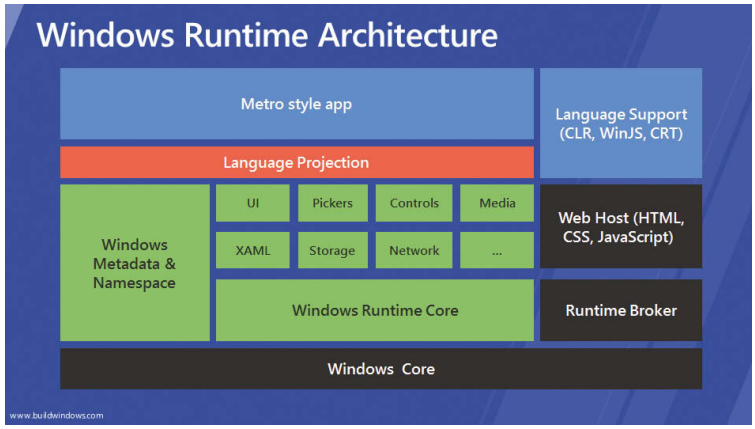
La prochaine étape dans cette convergence sera sans aucun doute la prochaine mouture de Windows. Patience toutefois, l'éditeur est pour l'instant totalement muet sur le sujet. Rendez-vous à la prochaine Build ?

Loïc **Rebours**
Consultant .NET chez Avanade
MVP Client Development bit.ly/ENIW81

Ecrire des composants WinRT en C++ avec WRL

Depuis la sortie de Windows 8, Microsoft a proposé un ensemble de nouveaux outils et de technologies pour le développement des applications « Windows Metro Style Apps ». Dans le monde Metro qui est l'ancien nom marketing de ces applications disponibles sur le Microsoft Store, on parle d'interfaces XAML, du Windows Runtime et de composants WinRT. Si vous êtes habitué au développement Windows traditionnel, vous connaissez les API Windows Win32. Dans le monde des « Windows Store Apps » qui est la nouvelle appellation de ces applications pour le Windows Store, tout se concentre autour d'une brique technologique qui est le Windows Runtime. Cette brique et son architecture ont été présentées à la conférence BUILD 2011 de Microsoft suite à la sortie beta de Windows 8.





Le Windows Runtime se compose de composants techniques et d'un ensemble d'API que sont les Windows Runtime APIs. La programmation des applications Windows Store Apps est faite de l'utilisation de ces APIs. Ces APIs sont disponibles sous formes de composants WinRT qui s'exécutent sur un socle Windows standard.

Les technologies des composants COM et OLE2

Historiquement, les composants COM sont une technologie conçue par Microsoft pour l'échange de composants binaires entre différents langages comme Visual C++ et Visual Basic. La technologie COM est à la base de la technologie OLE2 qui a aussi été nommée par un terme marketing plus général qu'est ActiveX. Les modèles COM et OLE2 fournissent plusieurs interfaces pour créer des composants plus ou moins complexes, exposer et échanger des propriétés, des méthodes, des évènements, des contrôles graphiques réutilisables et des mécanismes de communication. Les deux interfaces les plus connues de ce modèle de composants sont IUnknown et IDispatch. Les composants COM doivent être enregistrés dans la base de registre locale du poste.

Un composant WinRT c'est quoi ?

Pour Windows 8, Microsoft étend la technologie COM. En effet, un composant WinRT est un composant COM de nouvelle génération. Une nouvelle interface fait son apparition. Elle est disponible dans le fichier inspectable.idl. Cette interface hérite de IUnknown et fournit 3 nouvelles méthodes que sont Getlids, GetRuntimeClassName et GetTrustLevel :

```
interface IInspectable : IUnknown
{
  HRESULT GetIids(
    [out] ULONG * iidCount,
    [out, size_is(,*iidCount)] IID ** iids);

  HRESULT GetRuntimeClassName( [out] HSTRING * className);

  HRESULT GetTrustLevel([out] TrustLevel * trustLevel);
}
```

Dans le modèle COM traditionnel, les interfaces et composants sont décrits dans des fichiers IDL et sont compilés par le compilateur MIDL pour produire des fichiers d'entête .h et .c pour le monde C/C++ et des fichiers TLB (Type Library) pour les autres langages. Avec les composants WinRT, la logique est conservée.

On décrit les interfaces et les composants (appelés runtimeclass au lieu de coclass) dans un fichier IDL et on compile avec MIDL.

Le compilateur MIDL produit des fichiers .h et .c mais aussi des Windows metadata (.winmd) comparables à ceux qui existent dans le monde du .NET Framework. Il est possible d'ouvrir les fichiers d'extension winmd dans l'outil ILDASM pour explorer les interfaces, les classes, les méthodes et les propriétés.

Donc un composant WinRT, c'est un composant qui implémente IInspectable et qui possède des Windows metadata. C'est tout !

Ecrire un composant WinRT en C++

Il existe deux possibilités en C++ pour développer des composants WinRT :

- Utiliser le C++/CX
- Utiliser la librairie WRL (Windows Runtime Library) disponible dans le Windows SDK

Introduction à C++/CX

Le C++/CX est un ensemble d'extensions du compilateur Visual C++ pour écrire des composants WinRT. C++/CX possède une syntaxe et des mots clés que ne possède pas le C++ traditionnel avec des ^et des % comme le C++/CLI. Cependant le code généré est natif ; ce n'est pas du .Net comme le C++/CLI. Un composant simple en C++/CX ressemble à cela :

```
namespace MyWinRTComponent
{
 public ref class Logger sealed
 {
 public:
   Logger() { }
 public:
   void LogInfo(String^ message) { … }
 };
}
```

Il suffit de déclarer une classe avec les mots clés publics ref et sealed et toutes méthodes publiques seront exposées. Le but de cet article n'est pas d'illustrer C++/CX car ce dérivé de C++ n'est pas le vrai C++ ISO. Il est peut-être intéressant pour des nouveaux développeurs, mais son adoption n'est pas un franc succès et la communauté C++ est attachée au vrai C++, celui qui est défini comme un standard ISO. Il existe une autre possibilité qui est franchement plus intéressante. Microsoft l'utilise pour faire les composants qui sont livrés avec Windows 8. Cette autre façon consiste à utiliser la librairie WRL qui a pour but de couvrir tous les principes internes exposés par le Windows Runtime. Le Windows Runtime est fabriqué avec la librairie WRL alias *Windows Runtime Library*.

Utiliser la librairie Windows Runtime Library (WRL)

WRL est disponible dans le Windows SDK de Windows 8 et Windows 8.1 sous C:\Program Files (x86)\Windows Kits\8.x\Include\winrt\wrl.

Fichier	Usages principaux
async.h	Gestion des mécanismes asynchrones
client.h	Classes ComPtr, WeakRef
def.h	Internes
event.h	fonction Callback()
ftm.h	internes
implements.h	RuntimeClass, ModuleBase
internal.h	internes
module.h	Module
wrappers\corewrappers.h	HString, HStringReference

Cette librairie est construite sur le même modèle que son ancêtre ATL. ATL est disponible dans Visual C++ pour faire des composants COM. WRL est disponible pour faire des composants WinRT. Pour faire un simple composant, il faut d'abord créer un fichier IDL qui va décrire les interfaces et les composants. Voici un composant simple dans un fichier IDL :

```
// Library1.IDL
import «inspectable.idl»;
import «Windows.Foundation.idl»;

#define COMPONENT_VERSION 1.0

namespace Library1
{
 interface ILogger;
 runtimeclass Logger;

 [uuid(3EC4B4D6-14A6-4D0D-BB96-31DA25224A15), version(COMPONENT
_VERSION)]
 interface ILogger : IInspectable
 {
   [propget] HRESULT Name([out, retval] HSTRING* value);
   [propput] HRESULT Name([in] HSTRING value);
   HRESULT LogInfo([in] HSTRING value);
 }

 [version(COMPONENT_VERSION), activatable(COMPONENT_VERSION)]
 runtimeclass Logger
 {
   [default] interface ILogger;
 }

}
```

Chaque interface et chaque composant ou runtimeclass requiert un identifiant unique qui est un GUID. Un composant WinRT implémente une ou plusieurs interfaces. Ces interfaces exposent des méthodes et/ou des propriétés, mais ne peuvent exposer que des types Windows. Pour échanger des types compatibles avec les langages comme C# et VB.NET, il faut utiliser des interfaces fournies par le Windows Runtime pour qu'il puisse mettre en œuvre les projections pour chaque langage. Pour échanger des types simples comme int, long, float ou double, il n'y a pas de problèmes.

Pour le type chaîne de caractères, il faut utiliser le type HSTRING. Ce type sera mappé automatiquement sur le type String en C#. Ce type HSTRING remplace le type BSTR qui existait autrefois pour exposer des chaînes de caractères dans le monde Visual Basic et OLE2. Il existe une classe qui encapsule ce type Windows et qui se nomme HString. Elle permet de travailler avec des types wchar_t.

Un composant WinRT avec WRL

Maintenant que l'interface et le composant sont définis dans le fichier IDL, il faut fournir une implémentation du composant. Nous allons tirer parti des classes et templates de WRL :

```
#include «Library1_h.h»

namespace ABI
{
 namespace Library1
 {
   class Logger : public RuntimeClass<ILogger>
   {
     InspectableClass(L»Library1.Logger», BaseTrust)

   public:
     STDMETHOD(get_Name)(HSTRING* value) { return S_OK; }
     STDMETHOD(put_Name)(HSTRING value) { return S_OK; }
     STDMETHOD(LogInfo)(HSTRING value) { return S_OK; }
   };

   ActivatableClass(Logger);
 }
}
```

La classe se trouve dans le namespace ABI. C'est une convention Microsoft. La classe hérite de `RuntimeClass<ILogger>` et le template RuntimeClass prend en argument l'interface que nous avons définie. La classe du composant utilise deux macros :

- InspectableClass qui contient le nom du composant et fournit l'implémentation de IInspectable,
- ActivatableClass qui fournit la factory nécessaire à cette classe.

Comme tous les composants COM, les différentes méthodes retournent un HRESULT. Contrairement aux composants COM traditionnels, les composants WinRT n'ont pas besoin d'être enregistrés dans la base de registres. Ils doivent juste être accessibles au travers d'une class-factory. C'est la macro `ActivatableClass` qui fournit ce service via la classe `SimpleSealedActivationFactory`.

Il faut cependant un peu de code pour la plomberie WinRT. Si vous téléchargez le project template WRL, ce code sera ajouté automatiquement à votre projet WRL. Voici le fichier module.cpp qui ressemble à cela après quelques modifications mineures :

```
// module.cpp : Defines the module that contains the com classes
//
#include «pch.h»
#include <wrl\module.h>

extern «C» HRESULT WINAPI DllGetActivationFactory(_In_ HSTRING
 activatibleClassId, _Deref_out_ IActivationFactory** factory)
{
 auto &module = Microsoft::WRL::Module<Microsoft::WRL::InProc>
::GetModule();
 return module.GetActivationFactory(activatibleClassId, factory);
}

extern «C» HRESULT WINAPI DllGetClassObject(REFCLSID rclsid,
 REFIID riid, _Deref_out_ LPVOID *ppv)
{
 auto &module = Microsoft::WRL::Module<Microsoft::WRL::InProc>
::GetModule();
 return module.GetClassObject(rclsid, riid, ppv);
}

extern «C» BOOL WINAPI DllMain(_In_opt_ HINSTANCE, DWORD, _In
_opt_ LPVOID)
{
 return TRUE;
}

extern «C» HRESULT WINAPI DllCanUnloadNow()
{
 const auto &module = Microsoft::WRL::Module<Microsoft::WRL
::InProc>::GetModule();
 return module.GetObjectCount() == 0 ? S_OK : S_FALSE;
}
```

Les routines `DllGetActivationFactory` et `DllGetClassObject` font partie de la plomberie nécessaire pour WinRT.

Gestion des collections comme les vecteurs

Pour avoir une méthode qui retourne une collection d'objets comme un vecteur ou une map, le sujet devient beaucoup plus compliqué. En effet, Microsoft fournit un support étendu pour C++/CX et fournit un adaptateur pour faire le lien entre les containers de la STL et les interfaces Windows Runtime au travers d'un fichier d'entête nommé collection.h et disponible depuis C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC\include. Ce fichier se définit comme « `Windows Runtime Collection/Iterator Wrappers` ». C'est un ensemble de wrappers. Il va nous fournir des facilités pour passer du monde Windows Runtime au monde STL avec les containers STL comme std::vector<T>, std::map<K,T> et std::unordered_map<K,T>. Cependant, il utilise une syntaxe particulière au C++/CX. Dans notre cas, en véritable C++ ISO, il nous faut de vraies classes ISO C++. Nous devons étudier les interfaces du Windows Runtime à implémenter et fournir nos propres wrappers. Si nous voulons exporter une collection d'objets à base de std::vector<T>, il nous faut implémenter les interfaces suivantes présentes dans le namespace `Windows::Foundation::Collections` que sont `IVector<T>`, `IIterable<T>` et `IIterator<T>`.

Ainsi, du côté du consommateur du composant en XAML C#, par exemple, la collection IVector<T> est exposée en IList<T> par projection.

```
// C# client
Library1.Root root = new Library1.Root();
IList<String> list = root.GetVector();
foreach(string s in list)
{ ... }
```

En étudiant ces 3 interfaces ci-dessus, on se rend compte qu'il faut définir chaque méthode d'accès au container. Examinons les internes de IVector<T> qui sont définies dans le répertoire C:\Program Files (x86)\Windows Kits\8.x\Include\winrt et dans le fichier windows.foundation.collections.h. Ce fichier contient les différentes interfaces. Voici avec quelques modifications ce que représente IVector<T> :

```
template <class T>
class IVector : IInspectable /* requires IIterable<T> */
{
public:
 // read methods
 virtual HRESULT GetAt(unsigned index, T *item) = 0;
 virtual HRESULT get_Size(unsigned *size) = 0;
 virtual HRESULT GetView(IVectorView<T> **view) = 0;
 virtual HRESULT IndexOf(T value, unsigned *index, boolean
*found) = 0;
 // write methods
 virtual HRESULT SetAt(unsigned index, T item) = 0;
 virtual HRESULT InsertAt(unsigned index, T item) = 0;
 virtual HRESULT RemoveAt(unsigned index) = 0;
 virtual HRESULT Append(T item) = 0;
 virtual HRESULT RemoveAtEnd() = 0;
 virtual HRESULT Clear() = 0;
 // bulk transfer methods
 virtual HRESULT GetMany(unsigned startIndex, unsigned capacity,
 T *value, unsigned *actual) = 0;
 virtual HRESULT ReplaceAll(unsigned count, T *value) = 0;
};
```

D'après la documentation, l'interface IVector<T> hérite de IIterable<T> et sa représentation ressemble à cela :

```
template <class T>
class IIterable : IInspectable
{
public:
```

```
  virtual HRESULT First(IIterator<T> **first) = 0;
};
```

Et l'interface IIterable<T> retourne au travers de la méthode First() une interface IIterator<T> dont la représentation ressemble à cela :

```
template <class T>
class IIterator : IInspectable
{
public:
  virtual HRESULT get_Current(T *current) = 0;
  virtual HRESULT get_HasCurrent(boolean *hasCurrent) = 0;
  virtual HRESULT MoveNext(boolean *hasCurrent) = 0;
  virtual HRESULT GetMany(unsigned capacity, T *value, unsigned
  *actual) = 0;
};
```

Cette notion d'itérateur est semblable à l'itérateur du monde C++ de la STL. On possède un itérateur sur un container particulier. Le principe dans WinRT est le même. Maintenant que nous connaissons les interfaces et leurs méthodes, il faut les implémenter.

Nous allons définir un template construit autour du container STL std::vector<T> et faire le mapping entre l'interface IVector<T> et les méthodes de std::vector<T>. On remarque que la correspondance est assez naturelle pour la plupart des méthodes et ressemble presque à cela :

- IVector<T>.GetAt -> std::vector<T>.operator [index]
- IVector<T>.get_Size -> std::vector<T>.size()
- IVector<T>.InsertAt -> std::vector<T>.emplace(begin() + index, item)
- IVector<T>.RemoveAt -> std::vector<T>.erase(begin() + index)
- IVector<T>.Append -> std::vector<T>.emplace_back(item)
- IVector<T>.RemoveAtEnd -> std::vector<T>.pop_back()
- IVector<T>.Clear -> std::vector<T>.clear()
- IIterator<T>.get_Current -> *iterator
- IIterator<T>.get_HasCurrent -> bool
- IIterator<T>.MoveNext -> ++iterator

J'ai créé 2 templates pour implémenter ces 3 interfaces. Ces templates sont intrinsèquement liés à std::vector<T>. Le code de ces 2 templates est trop important pour figurer dans l'article, mais vous pouvez le récupérer en lien à la fin de cet article et aussi sur le Dev Center Windows Store App dans un code nommé « Windows Runtime Component using WRL and C++ for Async and Vector Collection ».

Voici la structure de ces 2 templates :

```
    template <typename T>
    class Vector : public RuntimeClass<IVector<T>, IIterable<T>>
    {
      InspectableClass(L»Library1.Vector», BaseTrust)

    public:
      Vector()
      {
        std::shared_ptr<std::vector<T>> ptr(new std::vector<T>());
        _v = ptr;
      }

…

    private:
      std::shared_ptr<std::vector<T>> _v;
    };
```

```
    template <class T>
    class Iterator : public RuntimeClass<IIterator<T>>
    {
      InspectableClass(L»Library1.Iterator», BaseTrust)

    private:
      typedef typename std::shared_ptr<std::vector<T>> V;
      typedef typename std::vector<T>::iterator IT;

      …

    private:
      V _v;
      IT _it;
      T _element;
      boolean _bElement = FALSE;
    };
```

Le template Vector<T> implémente les interfaces IVector<T> et IIterable<T> tandis que Iterator<T> implémente l'interface IIterator<T>. J'ai volontairement encapsulé le std::vector<T> dans un shared_ptr<T> pour les besoins de WinRT. En effet, il est possible de fournir une vue sur le vecteur et aussi un mode observateur sur le vecteur. C'est indispensable à implémenter si vous voulez supporter le binding.

Mais ce sujet dépasse le cadre de cet article. L'essentiel est d'avoir la mécanique qui permette de déclarer un vecteur de la STL et de l'exposer comme un type reconnu par WinRT. Maintenant que nous avons le wrapper pour retourner un IVector<T>, nous devons fournir un composant qui l'utilise et retourne la collection d'objets. Voici le composant Root qui possède deux méthodes : GetVector et GetVectorInt. Ces deux méthodes vont utiliser le même template pour retourner un IVector<T>. Voici la description IDL de Root et de son interface IRoot :

```
 [uuid(3EC4B4D6-14A6-4D0D-BB96-31DA25224A16), version(COMPONENT
_VERSION)]
 interface IRoot : IInspectable
 {
   HRESULT GetVector([out][retval] Windows.Foundation.Collections.
IVector<HSTRING>** value);
   HRESULT  GetVectorInt([out][retval]  Windows.Foundation.
Collections.IVector<int>** value);
 }

 [version(COMPONENT_VERSION), activatable(COMPONENT_VERSION)]
 runtimeclass Root
 {
   [default] interface IRoot;
 }
```

Voici la classe C++ du composant dans Root.h :

```
namespace ABI
{
 namespace Library1
 {
   class Root : public RuntimeClass<IRoot>
   {
     InspectableClass(L»Library1.Root», BaseTrust)

   public:
```

```
    Root() {}

  public:
    STDMETHOD(GetVector)(IVector<HSTRING>** value);
    STDMETHOD(GetVectorInt)(IVector<int>** value);
  };

  ActivatableClass(Root);
 }
}
```

Voici l'implémentation de la classe C++ du composant dans Root.cpp :

```
#include «pch.h»
#include «Root.h»
#include «Utility.h»
#include «Logger.h»
#include «Collection.h»

namespace ABI
{
 namespace Library1
 {
  STDMETHODIMP Root::GetVector(IVector<HSTRING>** value)
  {
   ComPtr<Vector<HSTRING>> v = Make<Vector<HSTRING>>();
   HString str;
   str.Set(L»String1»);
   v->Append(str.Detach());
   str.Set(L»String2»);
   v->Append(str.Detach());
   str.Set(L»String3»);
   v->Append(str.Detach());
   str.Set(L»String4»);
   v->Append(str.Detach());
   *value = v.Detach();
   return S_OK;
  }

  STDMETHODIMP Root::GetVectorInt(IVector<int>** value)
  {
   ComPtr<Vector<int>> v = Make<Vector<int>>();
   v->Append(10);
   v->Append(20);
   v->Append(30);
   v->Append(40);
   *value = v.Detach();
   return S_OK;
  }
 }
}
```

La fonction Make<> permet de créer un objet ComPtr<T>. On retourne cet objet avec la méthode Detach(). C'est la même mécanique qu'au temps de la librairie ATL. Il est possible de retourner d'autres types de collections que les vecteurs.

On peut retourner une map représentée par l'interface IMap<K,V> qui est exposée en projection au monde C# par IDictionary<K,V>. Cependant, il faut fournir le template qui « wrap » le container std::map<K,V>. Eh oui, il reste des choses à construire...

Les méthodes asynchrones

C'est un pattern largement utilisé dans les Windows Runtime APIs. Les méthodes async s'exécutent sans bloquer le thread d'appel. Dans le monde C#, l'utilisation d'une méthode async se fait avec l'utilisation du mot clé await. Nous allons créer un composant Root qui retourne un objet Logger au travers son interface ILogger. Voici le code client XAML C# client qui fait appel à notre composant WinRT :

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
 Library1.Root root = new Root();
 Library1.ILogger logger = await root.GetLoggerAsync();
 logger.LogInfo(«LogInfo()...»);
}
```

Comme pour la gestion des collections, la gestion des méthodes async est facile à réaliser avec C++/CX. En effet, la librairie PPL Taks a été modifiée pour supporter la création de tâche avec le pattern async. Ce pattern met en œuvre plusieurs interfaces du Windows Runtime. Pour le monde du vrai C++ ISO, la librairie WRL fournit la classe AsyncBase<T> qui fournit une implémentation de la machine à état nécessaire pour le fonctionnement des méthodes async. La mauvaise nouvelle est que AsyncBase n'est pas documentée... Il suffit de faire une recherche sur les mots « WRL AsyncBase ». Avec Bing, on obtient 14 résultats ; un record ! Pour tirer parti de cette classe, il suffit de réaliser une classe dérivée de la classe AsyncBase et de fournir une surcharge de certaines méthodes virtuelles. Pour fournir la mécanique d'exécution asynchrone, nous allons utiliser PPL (Parallel Patterns Library) et la class task<T>. Pour comprendre la mécanique async du Windows Runtime, il faut s'intéresser aux interfaces suivantes :

- Windows::Foundation::IAsyncOperation<TResult>
- Windows::Foundation::IAsyncOperationWithProgress<TResult, TProgress>
- Windows::Foundation::IAsyncAction
- Windows::Foundation::IAsyncActionWithProgress<TProgress>

Ces quatre interfaces héritent de Windows::Foundation::IAsyncInfo. Pour créer une méthode async, je vais utiliser un template qui tire parti de AsyncBase et qui exécute la fonction passée en paramètre dans une task de PPL. Voici à quoi ressemble le composant Root qui retourne un Logger de manière async :

```
STDMETHODIMP Root::GetLoggerAsync(IAsyncOperation<ILogger*>
** value)
{
 ComPtr<task_based_async_operation<ILogger>> pObject =
Make<task_based_async_operation<ILogger>>(
   std::async([&]() -> ILogger*
 {
   ComPtr<Logger> p = Make<Logger>();
   return p.Detach();
 }));
 *value = pObject.Detach();
 return S_OK;
}
```

Le template classe task_based_async_operation prend en paramètre l'interface de retour (ici ILogger) de la fonction async. Dans notre exemple, la task PPL qui est créée ne fait que construire un objet Logger sachant que cet objet implémente l'interface ILogger.

Code source des templates C++ présents dans cet article

Vous pouvez télécharger le code source des templates C++ sur le « Dev Center - Windows Store App » en faisant une recherche sur « Windows Runtime Component using WRL and C++ for Async and Vector Collection » ou bien taper l'URL suivante : http://code.msdn.microsoft.com/windowsapps/Windows-Runtime-Component-4dc6fa20

Ce code vous permet de démarrer avec un projet de dll Win32 pour WinRT.

Demandez plus aux équipes Microsoft US

J'ai eu quelques échanges avec l'équipe Visual C++ de Microsoft et je leur ai fait part de ma surprise de constater que seul le support de C++/CX est réellement bien documenté par le biais d'exemples en standard dans le produit Visual C++, mais, que concernant C++ et WRL, il n'existe toutefois qu'une description sommaire des classes. En effet, les développeurs de Microsoft utilisent la librairie WRL pour faire Windows 8 et ses composants WinRT; WRL peut être considéré comme le successeur de ATL. J'ai expliqué que j'avais eu du mal à trouver des exemples ou de la documentation sur IVector<T> et `windows.foundation.collections.h`. Pour la gestion des méthodes async par exemple, c'était la solitude complète... Bing m'a retourné 14 résultats sur AsyncBase. Pour la gestion des collections à base de IVector<T>, j'ai bien vu des choses dans le code de Mozilla Firefox pour Windows RT et celui de Google Chromium pour WinRT qui m'ont confirmé que ce n'est que de l'implémentation standard d'interfaces COM ; c'est navrant de devoir aller dans leur répository Git ou Svn pour trouver des indices sur les technologies Microsoft... Pour le moment, les échanges avec les US sont limités mais Microsoft est à l'écoute de la communauté C++... On m'indique que le code généré pour C++/CX est optimisé pour la performance. Oui, peut-être, mais écrire une classe en C++ avec WRL n'est pas très dur, et je peux utiliser du C++ ISO, ce qui est très rassurant plutôt que les extensions du compilateur Visual C++ et sa syntaxe ^ et % qui n'a rien à voir avec celle du C++ standard. Microsoft utilise WRL en interne et ça me rassure. Avec du code C++ standard, je peux réutiliser mon code existant, je peux le porter sur d'autres plateformes facilement en « wrappant » les templates et macros avec des coquilles vides. Bref, je ne me ferme aucune porte. Et puis j'avoue que j'ai un faible pour la technique donc je préfère maitriser mon code plutôt que d'être dépendant d'un compilateur. Le C++ donne le pouvoir et la performance. Si comme moi, vous trouvez que lire les entêtes du Windows Kits pour comprendre les templates C++ et les interfaces du Windows Runtime devrait être accompagné d'un peu plus de documentation et d'exemples de code, manifestez-vous et ils répondent. Voici les contacts à utiliser :

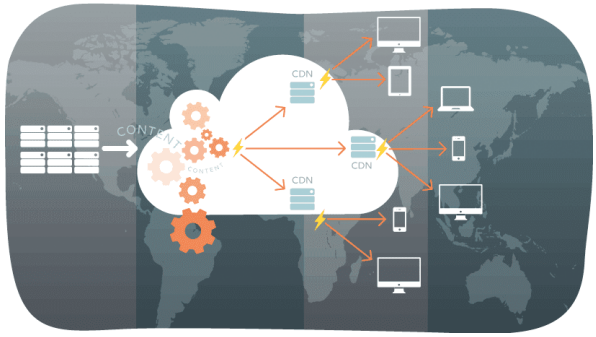
- Visual C++ General Manager : Alessandro.Contenti@microsoft.com
- Visual C++ Lead : Tarek.Madkour@microsoft.com
- L'auteur de WRL : Sridhar.S.Madhugiri@microsoft.com

Je dois vous avouer qu'il existe d'autres éléments à creuser comme par exemple la notion du binding, les collections observables, les events et les delegate. Bref, si vous êtes un programmeur C++ curieux, vous avez la possibilité de comprendre les internes du Windows Runtime rien qu'en regardant et en lisant les fichiers d'entêtes du Windows Kits qui sont des templates C++. Avouez que c'est bien plus fun que de faire du C# sans rien comprendre des couches sur lesquelles on s'appuie ? A méditer...

Christophe **Pichaud**
.NET Rangers by Sogeti
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com | www.windowscpp.net

Node.js : la tendance JavaScript côté serveur

Depuis plusieurs mois, JavaScript est le langage le plus actif sur github, loin devant Java ou PHP. Outre une forte présence sur le « Front » des applicatifs, JavaScript commence à gagner ses lettres de noblesse côté « Serveur ».



En témoignent aujourd'hui les nombreux projets autour de Node.js qui est de plus en plus choisi pour expérimenter des problématiques de temps réel ou de quasi-temps réel.

L'objectif de ces projets est de proposer des expériences utilisateurs dans des environnements très connectés (jeux vidéo, mobiles, sondage grand public, notification, etc.). À titre d'exemple, le chat et le moteur de recherche de Facebook sont développés en partie avec Node.js. L'application mobile de LinkedIn repose partiellement sur NodeJS.

Les premiers retours d'expérience montrent que cette technologie est encore jeune donc complexe à maîtriser, en particulier sur des projets en production, mais très prometteur. Elle a déjà trouvé un public et une communauté passionnée et dynamique.

Node.js pour des projets « temps réels », mais pas seulement...

V8 (le moteur JavaScript de Chrome) est au cœur de Node et il est très performant grâce à son compilateur JIT (Just In Time). Dans Node.JS, il est associé à libuv, une librairie asynchrone. On obtient alors un ensemble très performant capable de traiter simultanément des milliers de requêtes. En effet, la conception asynchrone de Node.JS évite une programmation itérative « bloquante » et limite les attentes d'entrée/sorties (filesystem, appels HTTP, etc ..). Ce modèle asynchrone / événementiel n'est pas nouveau, il est même au cœur du serveur Web Nginx, l'un des serveurs Web les plus populaires sur Internet.

Au-delà de la performance pure, c'est surtout le ratio performance/taux d'utilisation des ressources qui est nouveau. Il rend Node.js capable de traiter une volumétrie importante de requêtes sur des machines « normales ». À titre d'exemple, AF83, lors des élections présidentielles, a développé une solution de sondage en Node.js qui a permis de tenir 20 000 connexions à la seconde avec 1 seul processus attaché à un vCPU. De plus, la capacité de traitement de Node.JS peut facilement être répartie sur plusieurs vCPU grâce au mode cluster.

La réduction du temps de traitement et la capacité à traiter de nombreuses requêtes sont les critères majeurs qui amènent une équipe projet à choisir Node.js. Jusqu'à présent, il fallait utiliser du C, du C++ ou du Java pour se lancer dans de tels projets. Aujourd'hui, même si Node.JS ne permet pas de concurrencer ces langages sur certains besoins, il le met à portée de beaucoup d'équipes. Mais attention, faire des choses peu performantes en Node.JS arrive et il est tout à fait possible de coder des choses bloquantes sans s'en rendre compte.

Enfin, les librairies et modules exposés par l'écosystème Node.js facilitent la mise en œuvre d'API REST capable d'interagir dans des univers multi-devises : mobile, desktop, TV, etc. Cet avantage, couplé à un langage qui offre une portabilité côté serveur, navigateur ou tout terminal doté d'un moteur JS, fait du couple Node.js/JavaScript un choix de prédilection pour certaines applications multidevises ou Web.

Ses qualités

- Performances, ou plutôt capacité à gérer de très nombreuses requêtes simultanées.
- Modèle événementiel / asynchrone,
- Gestion native du HTTP et plus généralement de tous les protocoles réseaux (TCP/UDP)
- Multiplateformes (mobile, desktop, TV),
- Portabilité du code JavaScript aussi bien côté serveur que client,
- Une communauté très active.

Node.js nécessite une bonne maîtrise du langage JavaScript : exit les (Java)Script-kiddies.

Node.js, c'est du JavaScript. Or, la vraie compétence JavaScript est rare : savoir utiliser Jquery ne veut pas dire être compétent en JavaScript. Un développeur JavaScript compétent est à minima capable de faire de la programmation objet (à partir des prototypes), fonctionnelle et événementielle. La courbe d'apprentissage est donc longue, Node.js et JavaScript recèlent de nombreuses subtilités (scope, fonctions, this, événements).

La montée en compétence est vite consommatrice de temps. Pour les novices en programmation, il faut compter 4 à 5 mois pour commencer à maîtriser l'environnement.

Son passé associé à l'animation de page HTML l'a rendu impopulaire auprès des développeurs expérimentés. Mais, même si les a priori sont encore forts sur ce langage, il est normalement accessible pour des développeurs chevronnés. Après avoir expérimenté un premier projet, ils sont souvent convaincus et même très enthousiastes à propager la bonne parole !

Les limites de Node.JS

- la complexité de la programmation événementielle

Le modèle asynchrone est un concept très important : il est omniprésent dans le fonctionnement de Node.JS.

Même après quelques utilisations, on peut facilement tomber dans des pièges de conception asynchrone.

À titre d'exemple, même une fonction d'écoute de port est asynchrone :

```
var server = require('HTTP').createServer(function(req,res)
{res.end('hello')});
server.listen(8080, function (){console.log('listening on 8080')})
console.log('hello');
```

donnera

```
hello
listening on 8080
```

Cette programmation événementielle rend le débogage plus compliqué : il faut faire des callbacks, bien passer les contextes, gérer les exceptions... et surtout leur contexte. Pour la gestion des exceptions, 2 possibilités s'offrent aux développeurs : attraper toutes les exceptions et correctement les traiter ou simplement redémarrer. Chez Fasterize, toutes les exceptions sont attrapées sans que les process ne s'arrêtent mais par contre, ces exceptions sont ensuite remontées. Heureusement, pour faci-

liter la tâche, on retrouve des outils comme node-inspector basé sur le debugger JavaScript de Chrome (breakpoints, stacks, etc.).

◗ Une v1 qui se fait attendre

Nouvelle lib HTTP, SSL performant, cluster clean, execution des child process en mode synchrone et des tonnes d'améliorations tournées vers la performance, c'est peu de dire que cette nouvelle version est attendue (et depuis plus longtemps que les précédentes) !

◗ Trop de modules tuent le module

De 5000 à 65000 en deux ans ! Presque 300 nouveaux modules par jour ! L'écosystème des modules est fantastique, on trouve tout ce qu'on veut mais le revers de la médaille est qu'il faut chercher et trouver le bon module, à savoir un module testé, utilisé et maintenu.

◗ Windows ?

La très grande majorité des développeurs NodeJS sont sous Mac ou Linux et Windows fait étonnamment figure de parent pauvre. Les développeurs sous Windows galèrent régulièrement à cause de modules ou de fonctionnalités qui ne sont pas pensées pour leur OS ...

Architecture

Node.js exige de réaliser des applications simples et spécialisées pour maîtriser la complexité induite par l'asynchrone. Il faut concevoir une architecture logicielle adaptée à Node.js avec un découpage des actions complexes en applications simples et spécialisées. La philosophie est d'ailleurs assez proche d'Unix/Linux avec une multitude de petits programmes faisant une seule chose mais qui le font bien, avec une entrée, une sortie et qu'on peut chaîner entre eux (cf. awk | grep | sort | uniq). Par exemple, Fasterize a conçu son moteur de webperformance en SaaS en distribuant des tâches sur des workers asynchrones ou bien au niveau du proxy, en injectant des middlewares chacun responsable de la modification d'une partie de la requête.

En fait, assez rapidement, on en vient à créer ses propres modules et à les assembler. Comme dans tous les langages, la gestion des modules est quelque chose de complexe (dépendances), mais NodeJS a réussi dans la majorité des cas à en faire quelque chose de simple grâce à npm (Node Package Manager), l'outil de gestion des modules intégré à Node. Reste que cette gestion n'est pas parfaite et qu'il existe des débats sur l'utilisation de npm / git / node_modules.

En production, comme tout process server, il faut prévoir un mécanisme de surveillance. Chez Fasterize, vu l'immaturité des outils, tout le câblage classique a été redéveloppé : gestion des PID, logs, des services, etc.... Aujourd'hui, de nombreux modules existent pour ça.

Il faut encore être très vigilant sur les mises à jour de vos modules. La communauté enrichit (trop) rapidement l'éco-système de modules prêts à l'emploi et cela peut avoir un impact sur les performances ou même le fonctionnel. Il faut donc rester prudent sur la montée en version et avoir une politique rigoureuse de tests.

Scaler ses applications NodeJS n'est pas plus compliqué que dans les autres langages, il faut juste bien concevoir son application pour qu'elle soit distribuée, stateless et asynchrone. Par contre, grâce au module cluster, NodeJs peut scaler facilement d'une machine à une autre plus puissante et disposant de plus de cœurs CPU. Comme chaque NodeJS est monothreadé, il ne tourne par défaut que sur un seul cœur. Le module cluster permet de déployer le même process sur tous les cœurs de la machine. Par contre, ce module n'est pas encore nickel et la distribution sur les process n'est pas parfaite (une option round robin gérée par le Kernel arrive) et il faut gérer beaucoup de choses à la main (pas de graceful restart intégré par exemple).

Le debugging/profiling n'est pas évident à mettre en place mais commencent à apparaître des outils comme intégrés comme StrongOps après Nodetime ou TraceGL. En production, et NodeJs est magique pour ça, existe la possibilité de se connecter au process directement et d'inspecter tout ce qui a été exposé (node-repl), c'est extrêmement pratique pour comprendre ce qui peut se passer uniquement en production !

Déploiement d'un projet Node.JS

Pour déployer un projet NodeJS, il existe principalement deux manières : la première et la plus simple est d'utiliser un PaaS, la seconde est d'utiliser ses propres serveurs et de tout câbler soi-même.

Parmi les PaaS acceptant NodeJS, on peut citer : Heroku, NodeJitsu, CleverCloud, dotCloud, Azure, Engine Yard.

Chacune de ces plateformes permet de développer en local, de déployer très simplement (soit avec un git push soit avec un client spécifique) puis de configurer le lancement de ces applications.

Sans PaaS, il existe quelques outils de déploiement pour node mais aucun n'a franchement émergé, il faut donc souvent soit utiliser des outils non spécifiques à NodeJs, soit coder ses propres outils. Chez Fasterize, c'est Capistrano qui a été utilisé et adapté, car il avait déjà fait ses preuves dans les environnements RubyOnRails (possibilité de rollback, versionning des déploiements, etc. ...)

Bonnes pratiques

D'abord les bonnes pratiques que tout développeur doit respecter : mettre en place des tests unitaires. Les frameworks sont nombreux (mocha, sinon, should, etc.), fonctionnent très bien et sont bien documentés, c'est donc obligatoire !

NodeJS c'est de l'asynchrone et il est très facile de tomber dans le piège de «race conditions», même après des années de pratique. Dans les cas complexes, il peut être intéressant et utile de passer par des libs de de flow control comme async ou bien par les promises.

Ensuite, NodeJS c'est du JavaScript et il faut donc faire attention au problème de scope, de this, aux closures, aux erreurs dues au typage dynamique, etc .... C'est du Javascript qui tourne sur V8 et les développeurs ont souvent tendance à faire de la micro-optimisation contre-productive vu la façon dont le compilateur fonctionne. Il faut donc tester et mesurer pour prouver. D'ailleurs, c'est une constante : il faut mesurer le plus possible. Consommation mémoire pour détecter les memory leaks, temps passé dans la boucle événementielle pour détecter les blocages, etc. ... Par exemple, c'est comme ça que Fasterize a détecté des memory leaks dans la lib HTTP et dans la lib Gzip.

La communauté et la pérennité de Node.js

Beaucoup de personnes de la communauté prennent un véritable plaisir à expérimenter et construire autour du projet. Elle est incroyablement dynamique. Les modules sont arrivés très tôt et sont un véritable vecteur pour la communauté. Ils suscitent un fort engouement des développeurs qui dynamise le participatif : les évolutions sont donc très rapides et aussi maîtrisées. Par exemple, le développement du noyau est séparé des modules et ce qui confère une bonne stabilité au cœur de Node.js.

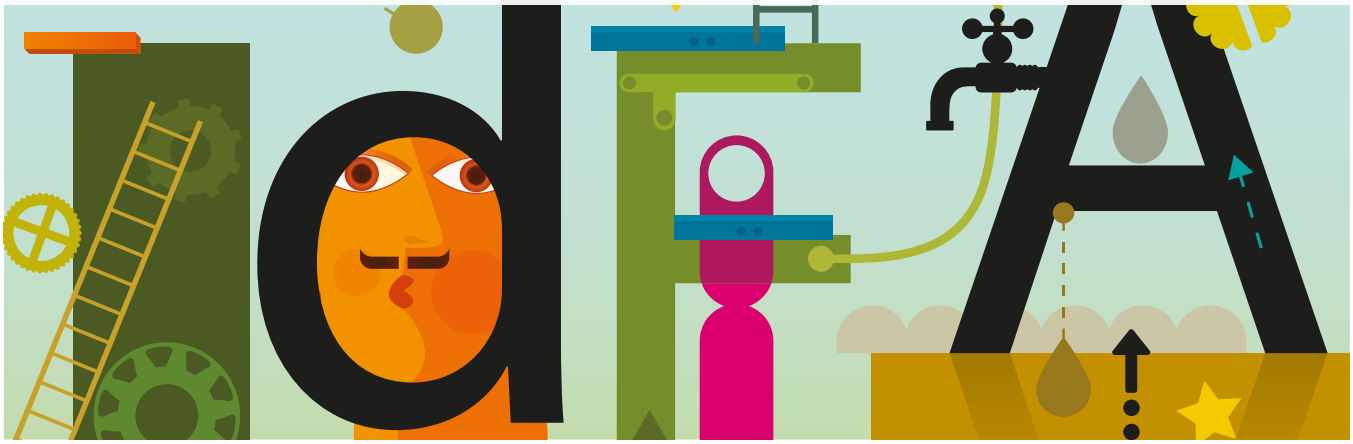
Aujourd'hui, Node.js est comme un nouveau continent à explorer, les contributeurs se permettent d'inventer ou réinventer des concepts. Il est peu étonnant qu'elle constitue l'une des communautés les plus actives sur GitHub. En revanche comme toute communauté naissante, elle se structure pour gagner en maturité et ces dernières semaines, les premiers drames entre acteurs importants de cette communauté sont apparus ...

Cet article a été réalisé à partir des retours d'expérience de Fasterize, d'AF83, SkillD et Toxicode durant l'Apérotech organisé par Oxalide dédié à cette thématique.

Sébastien Lucas, *CEO d'Oxalide*
Stéphane Rios, *CEO de Fasterize*

Découvrons les langages fonctionnels

Nous pourrions décrire un langage fonctionnel comme étant un langage dont les fonctions sont des objets de première classe. Avec cette définition, on aurait tout dit. Si nous cherchions plus loin, on nous parlerait de lambda calcul, d'effet de bord ou encore de système de type. Une telle présentation n'est pas très séduisante.



Dans notre expérience, l'intérêt pour nous de la programmation fonctionnelle découle de notre souhait et volonté de construire un logiciel fiable. Et si nous devions décrire un langage fonctionnel, nous parlerions de quasi-fiabilité et nous insisterions sur ce point. Cependant, il ne suffit pas de dire qu'un langage fonctionnel a des mécaniques qui rendent plus fiable votre développement, il faut le prouver.

L'effet de bord ou le démon de tout programmeur

La grosse difficulté dans le développement d'un logiciel réside dans notre capacité à visualiser tous les états possibles de notre programme (cette tâche peut largement se complexifier lors de l'utilisation de thread). Dans des paradigmes comme l'impératif ou l'objet, il semble naturel de faire des effets de bords. L'effet de bord, c'est ce qui peut faire brûler votre maison parce que vous incrémentez votre itérateur au mauvais moment. Plus concrètement, c'est le fait de manipuler des éléments externes à une procédure comme par exemple modifier une globale.

```
int global = 0;

int foo() { global += 1; return (global); }

while (foo() < 5)
  printf(«one tour\n»);
```

Dans l'exemple ci-dessus, on concrétise cet effet de bord. En effet, la fonction `foo` dépend désormais d'une globale. Il suffirait alors qu'une autre procédure change la valeur de `global` pour fausser dans cet exemple les 5 tours, sans que ce changement soit explicite dans le code de la boucle.
L'effet de bord implique donc un état implicite (car il n'est pas tout le temps représenté dans votre code) à votre procédure. Ainsi, il ne faut souvent plus réfléchir uniquement aux arguments de vos fonctions mais aussi à son contexte. Ce point, à un certain niveau, peut être source de bug. Nous dirons même qu'il est la source majeure de bug.

Programmation pure

Dans un langage fonctionnel, on parle de pureté du langage car on interdit tout simplement l'assignation d'une valeur à une variable. Ce point empêche drastiquement l'effet de bord dans notre programme car on considérera que toute variable est immutable.

```
let a = 21
let a = 42
(* ici, je n'assigne pas une nouvelle valeur à a, je crée une nouvelle variable qui va remplacer a *)
```

Ce point permet de considérer que nos fonctions peuvent avoir comme caractéristique la transparence référentielle. Cela veut dire que pour un même argument, la fonction retournera toujours le même résultat, qu'importe l'état de votre programme. La fonction ne dépend que de ses arguments, elle n'a pas d'effet de bord.
Des fonctions comme `malloc` ou encore `read` sont des fonctions à effet de bord car leurs procédures interagissent avec des éléments extérieurs (le *heap* ou encore le terminal). Au final, l'effet de bord est obligatoire sinon, nous ne pourrions pas interagir avec l'utilisateur. En cela, des langages comme OCaml mettent l'accent sur un style de programmation pur, tout en laissant la possibilité d'effet de bord.

Tout est fonction

La programmation pure ne vient pas de nulle part, son origine se trouve dans le lambda calcul. Nous n'allons pas théoriser ce système formel, il nous faudrait plus de pages. Ce qu'il faut s'imaginer, c'est qu'on puisse considérer un argument comme une fonction, qu'une fonction peut retourner une autre fonction, et que tout ceci aboutit à ce que l'on nomme la composition des fonctions.

```
let apply_operator f x y = f x y
print_int (apply_operator (fun a b -> a + b) 21 21)

(* je compose la fonction apply_operator avec une fonction anonyme qui additionne ses deux arguments *)
```

Cette technique permet de spécialiser le comportement d'une fonction selon ses arguments. Dans l'exemple ci-dessus, `apply_operator` va tout simplement exécuter la procédure `f` (en programmation fonctionnel-

le, on parle plutôt d'appliquer la fonction `f`) avec les valeurs `x` et `y`. Ensuite, on affiche le résultat de l'application d'`apply_operator` avec, comme argument, une fonction anonyme additionnant ces deux arguments et deux nombres.
Plus spécifiquement, en OCaml, on considère que tout est valeur. Puisqu'une fonction est une valeur (voilà pourquoi on dit qu'une fonction est un objet de première classe), OCaml va estimer que l'opérateur **+** est une valeur et donc une fonction. On peut factoriser le code ci-dessus de cette manière :

```
print_int (apply_operator (+) 21 21)
```

Enfin, OCaml intègre des mécaniques de la programmation fonctionnelle comme la *curryfication*. Celle-ci consiste en une application partielle de la fonction. Par exemple, nous pourrions créer une fonction prenant un argument et l'additionnant avec 3. Voici donc une équivalence de code :

```
let add_3 x = x + 3
let add_3 x = (+) 3 x
let add_3 = (+) 3

(* (+) 3 va tout simplement retourner une fonction attendant
un argument et l'additionnant avec 3 *)
```

Les trois lignes de code ci-dessus sont équivalentes, on fait place à la *curryfication* à la troisième ligne de code. Ainsi, le résultat de **(+)** 3 est une fonction (puisqu'on considère les fonctions comme des valeurs). On peut enfin faire de même pour `apply_operator` :

```
let add = apply_operator (+)
print_int (add 21 21)

let sub = apply_operator (-)
print_int (sub 21 21)
```

Mais nous pourrions au final aller plus loin en factorisant le code. Nous allons tout simplement remplacer `apply_operator` par son corps et factoriser une nouvelle fois.

```
let add x y = apply_operator (+) x y
let add x y = (+) x y
let add = (+)
```

Eh oui, nous avons redéfini l'opérateur **+** une nouvelle fois. Inutile mais indispensable pour comprendre concrètement le principe de la programmation fonctionnelle !

Conclusion

Nous avons vu les bases de la programmation fonctionnelle en montrant ses avantages et ses caractéristiques. Cependant, il y a encore nombre de choses à voir. Des langages comme OCaml ou encore Haskell vont plus loin, notamment avec leurs systèmes de type, mais aussi avec le *pattern-matching*. Ces fonctionnalités renforcent le caractère *safe* du langage car, au final, c'est bien de cela dont on parle. Les langages fonctionnels ont pour principal avantage d'offrir un développement *safe* au développeur. Qui n'a jamais rêvé de ne plus utiliser *valgrind* ou *gdb* ?

L'ÉTAT DE L'ART

C'est à se demander pourquoi on ne développe pas avec des langages fonctionnels. En réalité, les éminents C++ et Java s'empressent d'implémenter des traits fonctionnels dans leurs dernières versions. On pourrait presque parler d'une quête du Graal... Ceci ne faisant que souligner l'importance de la programmation fonctionnelle dans l'industrie. Cependant, cela n'est pas un effet de mode. La programmation fonctionnelle existe depuis bien longtemps (même avant le C avec le Lisp), et il est parfois curieux de voir certains principes de base des langages fonctionnels resurgir dans des domaines massifs de l'informatique. À croire qu'on y a baigné depuis tout petit.

Le cas JavaScript

Souvenez-vous de Netscape et Internet Explorer. Une guerre sans fin de parts de marché qui se réduisait à l'ajout massif de fonctionnalités. L'une d'entre elles était d'ajouter des effets à une page par le biais d'un langage. En temps de guerre, il s'agit parfois de faire des concessions, et la mode était au Java. Il fallait donc faire un langage ressemblant à Java (juste ressemblant). C'est donc en 1995 que Brendan Eich commença le projet en s'inspirant beaucoup de Scheme et de Self. Pour le premier de ces langages, c'est un dérivé de Lisp (langage créé par John McCarthy – le pionner de l'I.A., en 1958 au MIT) insistant sur l'aspect fonctionnel.
C'est alors que JavaScript est né, mais qu'on appelait encore DHTML (on se souvient tous des effets de neiges sur les pages web). Cela ne l'a pas empêché de renaître en 2000 notamment avec la technique AJAX. On a ensuite vu l'apparition de *jQuery* ce qui a considérablement démocratisé ce langage. Et nous avons aujourd'hui des projets tel que *Node.js* ce qui, au final, vaut à JavaScript la réputation d'assembleur du Web.
Mais derrière cet engouement se cache un langage fonctionnel !

```
var apply_function = function(f, a, b) {
  return (f(a, b));
};
```

Bien qu'ils n'aient pas toutes les mécaniques des langages tels OCaml ou Haskell, les projets ont pu tirer tous les avantages du paradigme fonctionnel. On peut notamment parler de la programmation par continuation en *Node.js*. Tout ceci n'est pas un hasard, les langages fonctionnels ont une bonne corrélation avec la programmation évènementielle.

```
function(res, req, done) {
  // ...
  done();
}
```

Son fondement a aussi pu faire naître des projets très intéressants tel que *js_of_ocaml*, un compilateur OCaml vers JavaScript. C'est tout de même un cas spécial dans le monde fonctionnel, de par son origine qui le place loin des instituts de recherche - qui sont souvent des nids à nouveaux langages fonctionnels - mais cette différence l'a justement propulsé en tant que référence aujourd'hui.

OCaml et l'industrie

OCaml est un langage fonctionnel qui a une longue histoire. En effet, il est né dans les locaux de l'Inria en France sous le tout premier nom de CAML par l'équipe Formel en 1987. La particularité de celui-ci était d'accorder une attention particulière au système de type. Le projet évolua avec le temps pour devenir Objective Caml en 1996 grâce à Xavier Leroy, Didier Rémy et Jérôme Vouillon.
C'est clairement un langage utilisé dans le domaine de la recherche tout particulièrement. Il n'en est pas moins incompatible avec l'industrie et le plus significatif des projets qui l'utilise est bien *Xen*.
Xen est un logiciel libre de virtualisation notamment utilisé par Amazon pour son service de cloud EC2 créé en 2002 à l'université de Cambridge.

Dans ce projet existe un logiciel du nom de *Xapi* qui est un outil de management du serveur *Xen*. Le problème était que le service *Xapi* ne devait pas crasher (au-delà d'autres contraintes tout aussi importantes telles que la taille du logiciel, les performances et l'intégration dans un environnement Unix). OCaml a clairement été une réponse à cette problématique notamment grâce à son attachement particulier au système de type, ce qui lui octroie une robustesse à toute épreuve.
Pour évoquer la puissance d'OCaml, le projet *Xen* est éloquent, surtout face aux problèmes techniques que sont les performances et la fiabilité (ce que contredisent souvent les détracteurs). Mais d'autres projets tout aussi impressionnants font vivre OCaml. On pourra citer le projet de Facebook *pfff*, ou encore *Astrée* du CNRS, l'ENS et l'Inria.
Nous pourrions relever aussi des entreprises n'utilisant qu'OCaml comme Jane Street (entreprise de micro-trading située à New York, Londres et Hong-Kong).

Et les jeux vidéo ?

Il y a encore un domaine dans lequel les langages fonctionnels n'ont pas pris place. C'est celui du jeu vidéo. Sur ce point, la critique ne se fait pas attendre, car on parlera de performance en particulier comme critère inaliénable à ce domaine (alors qu'aujourd'hui, on utilise *Unity*).
Il n'empêche que des références en développement de jeux vidéo conseillent la programmation fonctionnelle. En effet, cela n'a pas empêché Tim Sweeney, développeur d'*Unreal Engine*, valorisant Haskell comme étant une solution pour le développement de jeux vidéo – on pourra noter le jeu *Frag*, un clone de *Quake* en Haskell.
Dans la même veine, John Carmack, co-fondateur d'ID Software (entreprise de *Doom*, *Quake* et plus récemment *Rage*), conseille aussi la programmation fonctionnelle dans un de ses articles comme étant aussi un outil possible pour le développement de jeux vidéo.
Il est vrai qu'au final, il n'existe pas de jeu vidéo grand public utilisant un langage fonctionnel, mais ce n'est pas pour autant que la programmation fonctionnelle est fondamentalement incompatible avec ce domaine. Il peut même arriver que ce mode de conception puisse exposer de belles surprises.

Conclusion

La programmation fonctionnelle n'est pas un gadget ou une mode comme certains voudraient tant le faire croire. C'est une solution concrète à des problèmes concrets de l'industrie. En cela, il serait difficile de vous citer tous les domaines d'applications à la programmation fonctionnelle, les projets ne manquent pas, et nous vous invitons à les trouver.
Ce n'est pas non plus un modèle récent, et là où nous allons vous présenter des concepts *innovants*, votre serviteur vous dirait qu'ils existent depuis bien longtemps dans le domaine de la programmation fonctionnelle. Cela révèle surtout un retard du *mainstream* dans ce domaine, qui commence à reconnaître la qualité de ce paradigme (Java n'intègre la programmation fonctionnelle que depuis sa version 8).
Ce qu'il faut surtout en retenir, c'est qu'aujourd'hui, on ne peut plus ignorer ces langages qui font l'informatique d'aujourd'hui. Ils ont eu une ascension discrète mais forte qui n'a jamais laissé indifférentes les entreprises qui les ont adoptés. Et c'est en cela que nous vous invitons à entreprendre de vrais projets avec.

CAS CONCRET

La programmation fonctionnelle est présente dans nombre de domaines mais elle brille particulièrement dans un domaine qui est celui de l'analyse. En tant que professeur pour OCaml, nous cherchons souvent des projets montrant la puissance du langage et la facilité déconcertante à produire un logiciel qu'il serait plus éprouvant de produire dans d'autres langages, tel le C++. L'un des projets proposés dans le cursus Epitech est la *bistromatique*. Celle-ci consiste à créer une calculatrice à nombre infini. Au-delà de l'implémentation des routines d'addition, de soustraction, etc. Pour les nombres dépassant la taille d'un *int*, il y a aussi une problématique d'analyse d'une expression arithmétique pour la transformer en un arbre de syntaxe abstrait.
Celle-ci est souvent rencontrée dans le domaine de la compilation et de l'interprétation. Nous allons donc voir une implémentation en OCaml en parallèle à une implémentation en Python.

La Bistro, lexique

Le plus difficile dans l'analyse lexicale et syntaxique, c'est de se renseigner sur les techniques déjà existantes dans le domaine, et de les implémenter. La première phase de l'analyse consiste en une reconnaissance des mots composant une expression arithmétique. Ces mots définissent le lexique du langage (un peu comme un dictionnaire). Ainsi, on entame donc ce qu'on nomme une analyse lexicale qui, à partir d'une chaîne de caractères, en déduit une suite de mots ou plutôt de *tokens*.
Il s'agit de reconnaitre les caractères composant nos *tokens*, de compléter ces *tokens* et de délivrer ces fameux *tokens* sous la forme d'une liste.

```
type token =
  | Token_add | Token_sub | Token_mul | Token_div | Token_mod
  | Token_pal | Token_par
  | Token_number of char list
let lexing str =
  let ( >>= ) (cur_token, old_token, tokens) func = match old
_token with
    | Some old_token -> func None (cur_token :: old_token :: tokens)
    | None -> func None (cur_token :: tokens)
  in let ( >|= ) (digit, old_token, tokens) func = match old
_token with
    | Some (Token_number l) -> func (Some (Token_number (digit :
: l))) tokens
    | Some old_token -> func (Some (Token_number [digit])) (old
_token :: tokens)
    | None -> func (Some (Token_number [digit])) tokens
  in let rec aux old_token tokens idx =
    if String.length str = idx
    then List.rev @@ match old_token with
      | Some old_token -> old_token :: tokens
      | None -> tokens
    else match str.[idx] with
        | '+' -> ((Token_add, old_token, tokens) >>= aux) @@
(idx + 1)
        | '-' -> ((Token_sub, old_token, tokens) >>= aux) @@
(idx + 1)
        | '*' -> ((Token_mul, old_token, tokens) >>= aux) @@
(idx + 1)
        | '/' -> ((Token_div, old_token, tokens) >>= aux) @@
(idx + 1)
        | '%' -> ((Token_mod, old_token, tokens) >>= aux) @@
(idx + 1)
        | '(' -> ((Token_par, old_token, tokens) >>= aux) @@
(idx + 1)
        | ')' -> ((Token_pal, old_token, tokens) >>= aux) @@
(idx + 1)
        | '0' .. '9' as digit -> ((digit, old_token, tokens) >|
= aux) @@ (idx + 1)
        | _ -> match old_token with
```

```
        | Some token -> aux None (token :: tokens) (idx + 1)
        | None -> aux None tokens (idx + 1)
  in aux None [] 0
```

Voici donc le code OCaml qui va produire la liste de *tokens* en fonction de la chaîne de caractères. Dans ce code, nous définissons les opérateurs `( >>= )` et `( >|= )` localement à la fonction `lexing` qui vont tout simplement ajouter ou non le *token* à la liste selon `old_token`. On applique ensuite ce qu'on nomme le *pattern-matching*, qui va tout simplement regarder le caractère courant et faire l'opération en conséquence.
Si nous sommes face à un `'+'`, nous allons tout simplement ajouter `Token_add` après le `old_token` s'il y en a bien un en utilisant l'opérateur `( >>= )`. Enfin, nous avons le cas spécifique du nombre, qui va cette fois utiliser l'opérateur `( >|= )` pour regarder le `old_token`. En effet, si celui-ci est un `Token_number`, il faut, non pas rajouter le *token* dans la liste, mais compléter ce `Token_number` par le nouveau chiffre. Dans l'autre cas, nous ajoutons le `old_token` à la liste, et nous le remplaçons par un `Token_number` contenant qu'un seul chiffre.

```
class Token():
  add = 1
  sub = 2
  mul = 3
  div = 4
  mod = 5
  pal = 6
  par = 7
  number = 8

class Lexicon(dict):
  def key_list(self, keys, value):
    for key in keys:
      self[key] = value
def lexer(expr):
  lexicon = Lexicon({
    '+': Token.add,
    '-': Token.sub,
    '*': Token.mul,
    '/': Token.div,
    '%': Token.mod,
    '(': Token.pal,
    ')': Token.par
  })
  lexicon.key_list(
    ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'],
    Token.number)

  tokens = []
  datas = []

  for c in expr:
    if lexicon[c] == Token.number:
      if len(tokens) > 0 and tokens[-1] == Token.number:
        datas[-1] += c
      else:
        tokens.append(Token.number)
        datas.append(c);
    else:
      tokens.append(lexicon[c])
      datas.append(c)

  return (list(zip(tokens, datas)))
```

Voici l'équivalent en Python. La grosse différence entre les deux codes réside en 2 points.
Le tout premier est la facilité à créer un type en OCaml. En effet, en Python, l'apparition de l'énumération s'est faite tardivement, et elle n'égalera pas la puissance des *variants* en OCaml qui peuvent non seulement introduire un identificateur unique mais qui peuvent aussi s'accompagner d'une valeur (comme c'est le cas avec `Token_number` qui contient une liste de caractères). En Python, nous sommes obligés de jouer avec deux listes (ici `tokens` contenant les identifiants et `datas` contenant les valeurs).
Le deuxième point concerne le *pattern-matching* en OCaml (qui est une notion avancée de la programmation fonctionnelle) qui permet tout simplement un filtrage par motif. De l'autre côté, en Python, nous utilisons un dictionnaire pour associer au caractère un identifiant.

La Bistro, syntaxe

La deuxième phase consiste à transformer cette suite de *token*s en un arbre de syntaxe abstrait. Celui-ci est plus aisé dans son utilisation pour l'évaluation ou la décoration du code. Elle permettra de faire ce que l'on nomme une descente récursive dans l'arbre et d'appliquer une routine spécifique selon où l'on se trouve dans l'arbre.
Une représentation graphique semble être le plus approprié pour comprendre l'analyse syntaxique, voici un visuel d'exemple pour l'expression 2 + 3 * 5 : Fig.1.
Créer un nœud implique qu'on puisse savoir qu'il y a 2 *tokens* (l'opérateur et la deuxième opérande) plus loin du *token* courant (la première opérande). C'est en cela que nous sommes obligés de *sauver* les *tokens* dans une pile. Il s'agit d'ajouter nos nombres dans ce qu'on nomme *l'output* et de rajouter nos opérandes dans la pile. Une fois ceci fait, il suffira de dépiler 1 fois la pile et 2 fois *l'output* pour ensuite créer notre nœud. Dans le cadre de la gestion des priorités, nous allons tout simplement forcer cette action de dépilage selon l'opérateur et rajouter son résultat à *l'output* (et

Fig.1

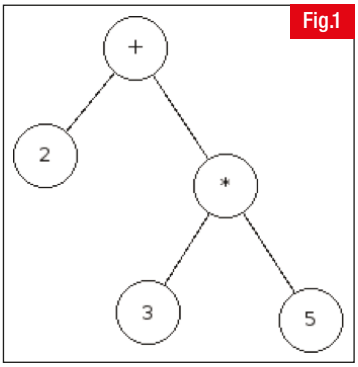
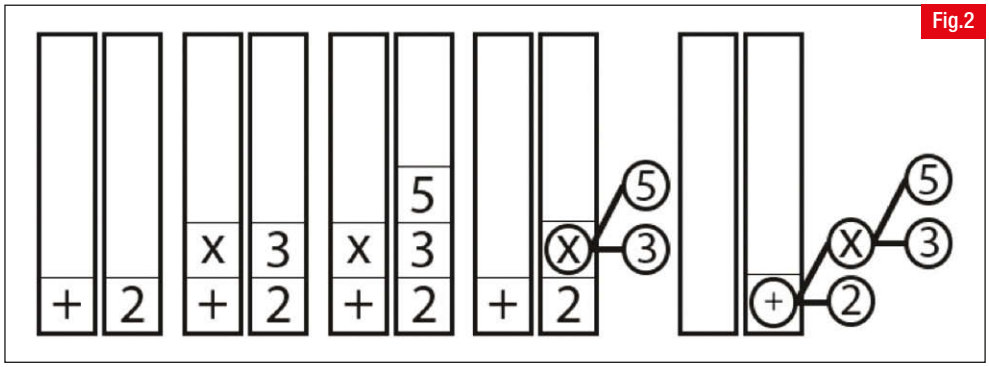


Fig.2



générer une partie de l'arbre). Voici un visuel de ce qui se passe séquentiellement avec l'expression 2 + 3 * 5 dans respectivement la pile et *l'output* : Fig.2.

```
type tree =
  | Add of tree * tree
  | Sub of tree * tree
  | Mul of tree * tree
  | Div of tree * tree
  | Mod of tree * tree
  | Number of int
let to_number lst =
  Number (List.fold_left (fun acc x -> acc * 10 + (int_of_char
x - 48)) 0 (List.rev lst))

let to_operator operator a b = match operator with
  | Token_add -> Add (a, b)
  | Token_sub -> Sub (a, b)
  | Token_mul -> Mul (a, b)
  | Token_div -> Div (a, b)
  | Token_mod -> Mod (a, b)
  | _ -> raise (Failure «Bad expression»)

let p = function
  | Token_add -> 5
  | Token_sub -> 5
  | Token_mul -> 10
  | Token_div -> 10
  | Token_mod -> 10
  | Token_pal -> 0
  | Token_par -> 0
  | _ -> raise (Failure «Bad expression»)

let parsing tokens =
  let node operator = function
    | b :: a :: r -> (to_operator operator a b) :: r
    | _ -> raise (Failure «Bad expression»)
  in let rec unstack o1 stack output = match o1, stack with
    | Some Token_pal, Token_par :: r -> r, output
    | Some Token_pal, o2 :: r -> unstack (Some Token_pal) r
(node o2 output)
    | Some o1, o2 :: r when (p o1) < (p o2) -> unstack (Some
o1) r (node o2 output)
    | None, o2 :: r -> unstack None r (node o2 output)
    | _ -> (stack, output)
  in let rec aux stack output = function
    | [] -> snd @@ unstack None stack output
    | (Token_add | Token_sub | Token_mul | Token_div | Token
_mod) as o1 :: r ->
      let (new_stack, new_output) = unstack (Some o1) stack
output in
      aux (o1 :: new_stack) new_output r
    | Token_par :: r -> aux (Token_par :: stack) output r
    | Token_pal :: r ->
      let (new_stack, new_output) = unstack (Some Token_pal)
stack output in
      aux new_stack new_output r
    | (Token_number l) :: r -> aux stack ((to_number l) ::
output) r
  in match aux [] [] tokens with
    | [ output ] -> output
    | _ -> raise (Failure «Bad expression»)
```

Le code est bien plus conséquent. Nous avons ici implémenté un automate à pile car il se caractérise par une `stack` dans son évaluation qui va justement sauvegarder nos *tokens* avant de créer notre arbre. Il s'agira simplement d'ajouter nos opérateurs dans cette pile, nos nombres dans `tree`, ce qui implique un *cast* explicite à l'aide de la fonction `to_number` (elle permet de passer d'une liste de caractère à un `int`), et de factoriser le tout avec la fonction locale `unstack` qui va créer notre nœud à l'aide de la fonction `node` et `to_operator`. Elle va donc dépiler une fois la pile et deux fois `tree` afin de créer un nouveau nœud.

```
class Stack(list):
  def push(self, item):
    self.append(item)
  def pop(self):
    return (self.pop())
  def is_empty(self):
    return (not self)

class Node():
  def __init__(self, data, left, right):
    self.right = left
    self.left = right
    self.data = data
def priority(token):
  return (({ Token.add : 5,
     Token.sub : 5,
     Token.mul : 10,
     Token.div : 10,
     Token.mod : 10})[token])

def parser(tokens):
  stack = Stack()
  output = []

  for (token, data) in tokens:
    if token == Token.number:
      output.append(Node(data, None, None))
    elif token in [Token.add, Token.sub, Token.mul, Token.div,
Token.mod]:
      for (op, op_str) in stack:
        if priority(token) < priority(op):
          a = output.pop()
          b = output.pop()
          output.append(Node(op_str, a, b))

          stack.pop()
        else:
          break
      stack.insert(0, (token, data))

  for (op, op_str) in stack:
    a = output.pop()
    b = output.pop()

    output.append(Node(op_str, a, b))

  return (output.pop())
```

La logique est quasiment la même en Python. La première différence, c'est que nous utilisons des traits impératifs dans le code (les actions telles que `append` ou encore `pop` sont à effet de bord car elles changent l'état de notre liste). Malgré le fait que les deux implémentent le même algorithme, on dénote encore des différences.
La deuxième différence concerne le *cast*. Il est plus aisé de *caster* en Python qu'en OCaml (la différence réside surtout dans le cast implicite et explicite). On pourrait dire que dans la concision du code, cela pourrait poser problème. Mais c'est bien une des différences qui nous pousse à continuer. C'est ce qu'on nomme un typage fort et dans le domaine de la programmation fonctionnelle, cette notion de typage est très présente avec OCaml mais aussi avec Haskell.
En effet, pouvons-nous considérer avec certitude que cette expression : `«Hello « + 42` donnera forcément `«Hello 42»` ? Nous vous dirions que tout dépend du langage. En C par exemple, nous n'aurons pas ce résultat. En réalité, ce résultat ne peut être qu'arbitraire.
Il l'est car il n'est pas correct au niveau du type. On en vient à se demander à quoi sert le type...
Le type permet de caractériser nos données. Le nom de la variable seul ne nous permet pas, ou difficilement, de caractériser une valeur. On utilise le type afin de définir non seulement la forme que peut avoir la valeur dans notre programme (un `float` est différent au niveau de la machine qu'un `int`) mais on définit aussi toutes les manipulations qu'il y a autour de cette valeur. Nous voudrions par exemple manipuler autrement notre liste contenant des bouts de nœuds de celle contenant nos opérateurs (et c'est le cas dans le code ci-dessus, l'une est une FIFO, l'autre est une LIFO). Il serait alors maladroit de notre part de mélanger les deux (et le résultat de notre *parser* en serait d'autant plus faux).
Malheureusement, qui nous dit en Python que nous mélangeons ces deux types de données ? L'exemple de la chaîne de caractères et du nombre est impossible en Python, cependant, créer une liste contenant des nombres et des chaînes de caractères est possible.
A la différence, en OCaml, on nous impose un typage fort, ce qui implique des *casts* explicites. Mais ceux-ci nous permettent de savoir exactement ce qu'il va se passer dans notre code (et de repérer l'erreur le plus tôt possible, il suffit de voir combien de fois nous levons une exception en OCaml avec `raise` et combien de fois on le fait en Python).

La Bistro, évaluation

La dernière phase de notre programme consiste en une descente récursive de l'arbre que nous venons de produire. Il s'agit donc de faire une fonction récursive qui, selon le type de nœud, va faire une addition, une soustraction, etc.

```
let rec eval = function
  | Add (a, b) -> ( + ) (eval a) (eval b)
  | Sub (a, b) -> ( - ) (eval a) (eval b)
  | Mul (a, b) -> ( * ) (eval a) (eval b)
  | Div (a, b) -> ( / ) (eval a) (eval b)
  | Mod (a, b) -> ( mod ) (eval a) (eval b)
  | Number i -> i
```

Comme vous pouvez le constater, à l'aide de la puissance du *pattern-matching*, le code est très simple en OCaml.

```
def eval(node):
  if node.data == '+':
    return (eval(node.left) + eval(node.right))
  elif node.data == '-':
    return (eval(node.left) - eval(node.right))
  elif node.data == '*':
    return (eval(node.left) * eval(node.right))
  elif node.data == '/':
    return (eval(node.left) / eval(node.right))
  elif node.data == '%':
    return (eval(node.left) % eval(node.right))
  else:
    return (int(node.data))
```

Voici l'équivalent en Python. Il faut savoir que Python gère déjà le calcul de nombres infinis ce qui fait qu'en l'état, ce code correspond à ce que voudrait le sujet de la *Bistromatique*. Pour OCaml, il faudra implémenter (mais là est aussi le défi) le type `Big_int` et le remplacer par le `int` présent dans notre type `tree`.

Conclusion

Pour l'objectif que nous avions à l'origine, à savoir implémenter une analyse des expressions arithmétiques afin de les évaluer, nous arrivons pour les deux langages à environ 100 lignes de code. Nous avons volontairement marqué les différences entre OCaml et Python, pour vous montrer l'intérêt de la programmation fonctionnelle, ce qui peut blesser quelques « *Pythonneux* »... Ce qu'il faut retenir, c'est qu'en l'état, on arrive très bien à faire des programmes en OCaml ayant quasi la même concision (même plus parfois) que ce que l'on peut produire en Python. Le *pattern-matching* et le système de type ne sont que des avantages à des logiciels bien plus conséquents. Cet ensemble cohérent qui forme OCaml nous offre une batterie de tests unitaires par le biais du système de type sans qu'on ait à rajouter une quelconque ligne de code.

CONCLUSION

Tout au long de ce dossier, nous avons donc appris la base de la programmation fonctionnelle en définissant ce qu'est une fonction, les effets de bord, et nous sommes allés jusqu'à voir la *curryfication*. Il faut savoir qu'aujourd'hui, la programmation fonctionnelle va plus loin notamment avec le *pattern-matching*, le système de type, mais aussi la programmation par continuation, ou encore la programmation monadique. Il y a donc encore beaucoup de choses à voir concernant généralement la programmation fonctionnelle. Ensuite, nous avons mis en évidence certains projets et langages qui font la programmation fonctionnelle d'aujourd'hui, comme JavaScript, mais en particulier OCaml avec Xen. Ils participent encore activement à l'industrie informatique, notamment avec le dernier projet en date de Facebook et son langage *Hack* dont une partie du compilateur (le synthétiseur de type) est faite en OCaml. On remarquera au final l'implantation généralisée de la programmation fonctionnelle, qui n'est plus réservée aux instituts de recherche.
Enfin, nous avons terminé par un cas concret d'analyse en reprenant un sujet d'Epitech pour montrer qu'il est viable de faire des programmes dans un langage fonctionnel (ici OCaml), et que ceux-ci peuvent largement rivaliser avec leurs conjoints C++ et Java.
Ce qu'il faudrait retenir de cette présentation, c'est l'importance qu'on accorde aujourd'hui à la fiabilité de nos logiciels et à l'expressivité de nos langages. Sur toute notre expérience, nous avons gagné la conviction que la programmation fonctionnelle a un réel avenir dans l'industrie (car elle correspond bien souvent à ce que l'on recherche), avenir qui ne se fera pas attendre longtemps. Cela a déjà commencé avec des projets d'une très grande envergure. Ce domaine n'est clairement plus une niche spécifique, c'est une réponse à des problématiques concrètes et il serait dommage que vous ratiez le cap. Alors commencez l'aventure !

Romain **Calascibetta** - *Epitech*

Linux : le poste de developpement ideal

Linux est un système adoré par certains et haï par d'autres. Ce sujet anime les forums depuis de nombreuses années. Qu'est-ce qui pousse les développeurs du monde entier à défendre et à œuvrer au bon fonctionnement de ce système ? Pourquoi tant de grandes entreprises lui font confiance ? Est-ce un système réservé à de sombres geeks développant des projets dont l'utilité n'est comprise que par eux seuls ? Pour comprendre ses avantages et inconvénients, il est nécessaire d'analyser les raisons pour lesquelles ce projet domine une grande partie du monde informatique.

A • Qu'est-ce que Linux ?

Pour beaucoup, Linux est un concurrent plutôt sympathique de Windows. Il est doté d'une interface graphique plus austère que ce dernier, et ne permet pas l'utilisation de logiciels bien connus de tous comme Ms Office ou autres logiciels de graphisme. Rares sont ceux qui savent qu'ils ont et utilisent au quotidien le système Linux sans même s'en rendre compte. On le retrouve dans les boxes ADSL, les téléphones Android, une grande majorité des sites Internet, lecteurs MP3, télévisions, GPS, pico ordinateurs, etc. Linux est un système très souple, capable de s'adapter aussi bien sur des ordinateurs ayant peu de puissance que sur de supers calculateurs. Cette capacité est en grande partie liée à son noyau, que l'on nomme plus couramment le Kernel.

B • Le Kernel

Le Kernel Linux est le cœur de tout le système. Linux désigne en réalité uniquement cette partie du système. C'est en 1991 qu'un jeune étudiant finlandais du nom de **Linus Torvalds** publie les toutes premières versions du noyau Linux. A cette époque, il était loin d'imaginer que son petit hobby allait prendre une telle ampleur dans le monde de l'informatique. 23 ans plus tard, Linus veille toujours sur les améliorations apportées au Kernel. Le noyau de Linux est la partie du système permettant aux différents périphériques et composants d'un ordinateur de dialoguer. C'est la couche basse qui permet d'exécuter une application, d'utiliser la mémoire, d'enregistrer des fichiers sur un disque dur ainsi que plein d'autres fonctionnalités essentielles au bon fonctionnement d'un ordinateur. Ces opérations sont très obscures pour bon nombre d'utilisateurs. A lui seul, le Kernel Linux n'aurait jamais eu un tel succès sans le système GNU.

C • Le système GNU

Le Système GNU a été conçu par une autre figure emblématique de l'informatique : **Richard Stallman**. Son but était de disposer d'une alternative libre à Unix. En 1990, après 7 années de travail, le système contenait déjà beaucoup d'outils permettant le développement d'applications telles que l'éditeur de code Emacs, les librairies en C ou le compilateur GCC. Le jeune Linus Torvalds était également l'un de ces développeurs qui utilisaient déjà ces outils, dont le compilateur GCC. L'utilisation de ces outils a été décisive dans le choix de faire de Linux un projet Open Source.

D • Interfaces graphiques

Pour rendre un système d'exploitation intuitif, il est nécessaire que ce dernier possède une interface graphique. Tous les utilisateurs d'un système ne sont pas forcément des administrateurs systèmes ou des développeurs qui se réjouissent avec un terminal et des lignes de commandes. Linux s'est très rapidement doté d'une interface graphique n'ayant rien à envier à OS X ou Windows.

On retrouve donc deux grandes familles d'interfaces graphiques :

- **KDE :** elle est réputée pour sa qualité graphique. C'est l'interface graphique idéale pour les utilisateurs de logiciels graphiques. On la retrouve souvent dans les studios de post-production audiovisuelle. Beaucoup de logiciels de graphisme utilisent la librairie graphique QT qui est à la base de l'interface KDE. La version 4.5 amène KDE à maturité. Stable et dotée d'une excellente gestion d'écrans multiples, c'est l'interface qu'il vous faut si votre domaine d'expertise porte sur l'imagerie, le montage vidéo ou la PAO.
- **Gnome :** Gnome est une interface graphique très appréciée des développeurs. Elle est plus légère que KDE et reste très conviviale. Son interface est très bien conçue pour changer d'application rapidement, ce qui est très appréciable en développement, au vu du nombre d'applications nécessaires pour concevoir des applications modernes. Si tel est votre cas, la version 3 de Gnome vous comblera.

redhat

debian GNU/Linux

D'autres interfaces graphiques existent, comme les projets LXDE et XFCE. On peut également citer Unity, l'interface graphique d'Ubuntu.

E• Distributions

Les distributions Linux sont des assemblages d'applications et de modules autour du noyau Linux. Elles permettent d'avoir un ensemble cohérent tout en facilitant son installation. L'aspect Open Source a permis à Linux d'avoir pour une même problématique, plusieurs solutions possibles. Pour les postes de travail, on compte trois grandes familles :

- **Debian :** Les Debian sont des distributions très réputées pour leur stabilité. On les retrouve sur beaucoup de serveurs. Certaines distributions dérivées comme Ubuntu et Mint sont ensuite apparues.
- **Redhat :** Les Redhat sont plus avancées en matière de versions de librairies que les Debian. Cela a pour avantage d'avoir les derniers correctifs plus rapidement, mais peut aboutir à plus d'instabilité. Ces distributions utilisent des dépendances RPM.
- **Suse :** La distribution Suse est un dérivé de Slakware. Suse avait très tôt proposé une solution entreprise garantissant un très bon support, et une très bonne compatibilité matérielle.

LA LOGIQUE DU SYSTEME

A• La philosophie

La philosophie de Linux est très liée à celle de GNU puisque le système se veut Open Source. Le concept de logiciel libre est très proche des communautés scientifiques.

On y retrouve donc des valeurs communes comme le partage de connaissances, une ouverture d'esprit ainsi que le fait d'aller à l'encontre des principes préétablis. Même si le concept d'Open Source n'est pas synonyme de gratuité, beaucoup de logiciels sous Linux sont distribués gratuitement.

B• Légèreté du système

Bien que le Kernel Linux gagne en fonctionnalités, cela n'impactera pas forcément la réactivité de votre système. Son noyau est une structure monolithique modulaire : j'entends par là que les différents modules sont contenus dans le noyau, mais seule une partie est active. Ce concept assure à Linux une intégration plus simple des fonctionnalités.

Les différents modules fondamentaux du noyau sont assemblés lors de la compilation. Il est donc possible d'ajuster les fonctionnalités de Linux et de compiler un noyau beaucoup plus léger afin de mieux répondre à ses besoins. Cette opération requiert certaines connaissances et surtout du temps.

Le fait d'avoir accès au code source du système nous permet de modeler Linux pour l'adapter et l'optimiser pour les tâches qu'il aura à traiter.

C• Tout est fichier

Dans Linux, tout est fichier (processeur, mémoire, disque dur, etc.) : c'est l'une des raisons pour lesquelles tant de développeurs l'ont adopté.

Ce principe peut paraître étrange à première vue mais il vous permet de récupérer facilement des informations sur le système.

Par exemple, si on lit le fichier /proc/cpuinfo, celui-ci contient toutes les informations du processeur : Fig.1.

On voit ici que les informations remontées par le système sont très détaillées, ce qui permet à des logiciels d'en tirer parti et ainsi de proposer des fonctionnalités très riches.

D• Un logiciel ne fait qu'une seule tâche, mais la fait bien

Contrairement à d'autres systèmes, les logiciels issus du système GNU/Linux ne réalisent qu'une seule et unique tâche.

Ces logiciels ont pour but d'être les plus optimisés possible et de traiter cette tâche au mieux.

On peut citer d'excellents logiciels comme :

- **cat :** il permet d'afficher le contenu d'un fichier,
- **wget :** un incontournable pour effectuer des requêtes http,
- **tail :** il permet d'afficher le contenu d'un fichier au fur et à mesure que ce dernier est modifié,
- **grep :** il a la capacité à rechercher du texte dans une arborescence de fichiers,
- **find et locate :** ils ont pour fonction de rechercher des fichiers. Pour locate, cette recherche s'effectue sur une base indexée,
- **top et htop :** ils donnent des informations sur les processus en cours d'exécution,kill : elle représente à elle seule un motif suffisant d'utiliser Linux ! Cette commande permet de tuer un processus, de gré ou de force avec différents niveaux de brutalité. D'autres systèmes n'ont pas cette capacité à forcer l'extinction d'un processus. Les développeurs ont souvent des serveurs récalcitrants : il est donc très pratique de ne pas devoir attendre que ce dernier réponde pour arrêter son processus.
- Etc.

E• Séparation du cœur et de l'interface

Alors que Microsoft a abandonné la séparation entre le cœur du système et son interface graphique depuis 2001, on est en droit de se demander pourquoi Linux conserve cette particularité. En fait, cela lui permet d'avoir des versions serveurs très optimisées. Peu d'utilisateurs se soucient de cette séparation, jusqu'au jour fatidique où ils travaillent sur un document très important et là... tout se fige. Que faire ? Redémarrer la machine violemment en espérant que cela n'aura pas corrompu notre précieux document ? La séparation du cœur du système et de son interface graphique a un véri-

Fig.1

```
user@computer# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Xeon(R) CPU E3-1225 V2 @ 3.20GHz
stepping        : 9
cpu MHz         : 3192.921
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon
pebs bts rep_good xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx
est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 x2apic popcnt aes xsave avx f16c rdrand lahf_lm ida
arat tpr_shadow vnmi flexpriority ept vpid
bogomips        : 6385.84
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:
...
```

table intérêt : il est en effet possible pour un collaborateur de se connecter à votre machine via le protocole SSH pour éteindre le processus responsable. Vous avez également la possibilité de relancer l'interface graphique.

F• Configuration centralisée

Derrière l'ensemble de Linux, on retrouve la configuration système du chemin /etc. On n'a donc aucun mal à retrouver un fichier de configuration. Bien que le panneau d'administration puisse offrir une aide, lorsqu'on a besoin d'administrer une machine, il est bien plus rapide de modifier directement le fichier décrivant le paramétrage. On retrouve par exemple dans le /etc, l'ordonnanceur de démarrage et l'arrêt de la machine sous forme de différents répertoires (rc0.d à rc6.d). On retrouve également dans le dossier /etc un fichier très important qui est le fichier profile : c'est ce dernier qui permet de définir les variables globales du système.

G• Sécurité

Linux a la réputation d'être très sécurisé. Cette réputation est en grande partie justifiée par l'expérience de ce système en matière de serveurs à travers le monde.

La communauté GNU/Linux ainsi que les plus grands acteurs du monde numérique travaillent pour détecter et corriger les failles. Lorsqu'une faille est détectée, un correctif est souvent disponible dans la journée.

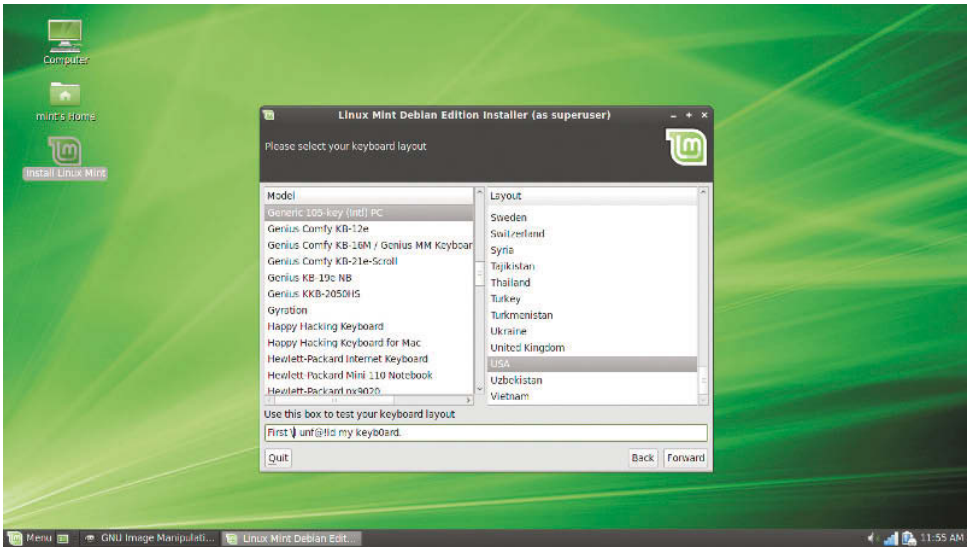
H• Firewall

Tous les systèmes GNU/Linux intègrent un pare-feu nommé iptables. Par défaut après l'installation il ne contient pas de règle de blocage. Il est donc nécessaire de le configurer suivant ses besoins. Cette opération est très simple via la commande « iptables ». Si on souhaite ajouter une règle dans le pare-feu, il suffit de taper ceci dans un terminal : Fig.2.

Comme pour toutes configurations sous Linux, la définition du pare-feu peut être automatisée via des scripts. Il existe également une interface graphique du nom de Firestarter qui permet aux utilisateurs peu habitués aux lignes de commande de configurer leur pare-feu.

L'HISTORIQUE DU GEEK

Linux a l'image d'un système conçu et utilisé par des Geeks férus d'informatique, que ce soient ses créateurs ou les développeurs contri-




```
root@computer:$ iptables -A (INPUT ou OUTPUT) -p (protocole) --dport (port) -j (ACCEPT ou
REJECT )
```

Fig.2

buant à son amélioration. Linux s'est simplifié et est devenu bien plus accessible. Pendant plusieurs années, il était d'ailleurs indispensable d'avoir de bonnes bases en informatique pour se débrouiller et utiliser correctement Linux. Le scripting Shell était une base obligatoire à tout développeur souhaitant concevoir des applications sous Linux. Ce dernier a été pendant très longtemps le système idéal pour tout développeur C/C++ : on y retrouvait des outils comme le compilateur « GCC » et « make » permettant de scripter la compilation. Ces outils sont encore couramment utilisés.

Très tôt, Linux a intégré le langage Python. Ce langage orienté Objet a démontré sa puissance dans le développement de scripts pour l'administration systèmes, d'applications Desktop, ainsi que pour des applications Web. On retrouve dans beaucoup d'applications OpenSource C++ des plugins en Python. Ces trois langages (Shell, C++ et Python) représentent les piliers de Linux. Le noyau et la plupart des applications Linux étant OpenSource, beaucoup de développeurs ont vu là le moyen de s'exprimer et rejoindre une grande communauté. Les projets GNU/Linux restent les seuls ayant réussi à allier un très grand nombre de développeurs à travers le monde.

LES OUTILS

A • Les terminaux

Le terminal reste l'outil incontournable de Linux. Il est possible à travers ce dernier de manipuler l'ensemble du système et d'exécuter bon nombre de programmes. Il repose sur différents langages de scripts (Shell, Bash, Zsh, etc...). Linux permet de chaîner les commandes : cette capacité, ainsi qu'un langage de scripts, permettent d'automatiser beaucoup d'actions. Linux compte énormément de commandes et de raccourcis clavier. En voici quelques-uns :

- **pwd :** permet de connaître l'emplacement actuel,
- **ls, ls -a, ls -l :** qui permet d'afficher les fichiers contenus dans le dossier courant. L'option -a permet d'afficher également les fichiers cachés. Pour l'option -l cela affiche les droits (lecture, écriture et exécution) des fichiers,
- **cd :** permettant de se déplacer dans les dossiers,
- **!! :** permet de répéter la dernière commande,
- **ps -ef|grep {nom d'un processus} :** cette commande permet de rechercher un processus parmi tous ceux en cours d'exécution,
- **top :** affiche les différents processus en cours ainsi que leurs consommations mémoire, cpu, etc.
- **netstat -lupat :** permettant d'afficher l'ensemble des processus connectés vers Internet,

- **chmod :** permet de changer les droits d'un fichier,
- **sudo :** permet d'exécuter un programme en tant qu'administrateur.

Il existe sous Linux différents logiciels permettant d'avoir accès à ces commandes qui répondent à des préférences ergonomiques. Pour la majorité d'entre elles, on retrouve de multiples onglets à thème graphique modifiable. Cela permet d'identifier rapidement le sujet de l'onglet sur lequel on travaille. Certaines consoles comme Terminator permettent de subdiviser les onglets afin d'avoir une vision plus détaillée et plus rapide des différentes actions en cours. Elle a également la capacité à synchroniser les commandes entres différentes parties subdivisées : lorsqu'on tape une commande dans une partie, une autre effectuera exactement la même action. C'est un avantage considérable dans un contexte avec différents environnements au sein desquels on souhaite effectuer les mêmes actions (mise à jour, déploiement d'application, etc.). Sachez que Linux intègre nativement Python 2.x. Ce langage est plus lisible et maintenable que le Shell.

B • Environnement de développement

Les IDE

- Les environnements de développement modernes ne sont pas très différents de ce qu'on retrouve sous Windows. La plupart des développeurs sous Linux utilisent Eclipse. Cet IDE a l'avantage d'avoir énormément de plugins qui lui permettent de prendre en charge la plupart des langages (Java, PHP, C++, Python, LaTeX, etc.). Sa communauté est très active puisqu'Eclipse est un projet Open Source distribué gratuitement. Cet IDE n'est pas exempt de défauts : il est assez consommateur en ressources !
- **Netbeans** est également très présent : c'est le concurrent direct d'Ecplise. Développé par Oracle, Netbeans présente des avantages en ce qui concerne l'intégration de Maven, comme une plus grande stabilité. Cependant, il a moins de plugins et sa communauté est plus restreinte.
- **IntelliJ** est l'un des meilleurs IDE à l'heure actuelle. Contrairement à Eclipse et Netbeans, il n'est pas Open Source et est payant (comptez 450$). Il présente cependant la meilleure intégration de Maven et d'outils comme Git. Il gère cependant moins de langages (Java, PHP, Python, Ruby, SQL).
- **Sublime Text** est un IDE très puissant, léger avec beaucoup de plugins. Il est développé en Python, ce qui rend la création de plugins très simple. Il n'est pas Open Source et payant, mais son tarif est très abordable (environ 70$).
- **KDevelop** est un très bon IDE pour les projets C++ et Python. C'est l'un des meilleurs IDE pour Python. Il a une intégration parfaite de ce langage. Il présente toutes les fonctionnalités nécessaires au développement d'un projet pour entreprise.

Le build process

- Beaucoup de langages modernes se basent sur **Maven** pour effectuer leurs compilations et packaging. On retrouve également cet outil sous Linux : ce dernier est bien plus utilisé en ligne de commande du fait du plus grand contrôle.
- **Make** est l'un des outils les plus répandus sous Linux pour la compilation. Il est principalement utilisé pour la compilation d'application C++ en association avec GCC. L'avantage de Make est d'être très proche du shell, ce qui en fait un très bon candidat pour l'intégration d'outils externes ou d'autres langages. On l'utilise également beaucoup pour la compila-

tion LaTeX. LaTeX est un langage permettant l'édition de documentation technique ou scientifique. Beaucoup d'applications Linux sont distribuées avec les sources C++. Lorsqu'on essaye des applications très récentes, on devient rapidement habitués à l'utilisation de make, la commande «./configure ; make; make install» permettant de configurer les librairies d'une application, la compiler puis l'installer sur le système.

C • Logiciels

Beaucoup de nouveaux Linuxiens se sentent un peu perdus sans les logiciels qu'ils connaissent sous d'autres systèmes. Cela se comprend, mais on retrouve bien souvent un équivalent sous Linux. L'inverse n'est pas forcément le cas. Beaucoup de logiciels sous Linux n'existent pas sous Windows ou sont des portages plus ou moins bien réalisés. Certains sont incontournables :

En graphisme :

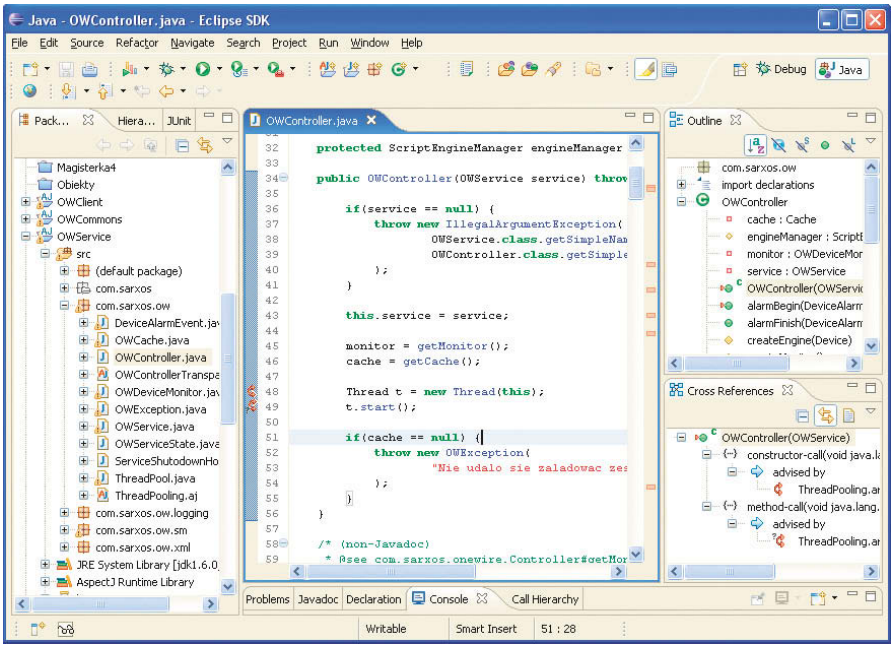
- **Gimp :** Un "Photoshop Like" Open Source. Ce petit logiciel donne l'impression qu'il est trop minimaliste aux premiers abords. Il offre pourtant presque les mêmes fonctionnalités.
- **Inkscape :** Un logiciel de dessin vectoriel simple et puissant.
- **Agave :** Un petit utilitaire très pratique, qui permet de donner des séries de couleurs assorties (complémentaire, monochromatique, triade, etc.).

En développement :

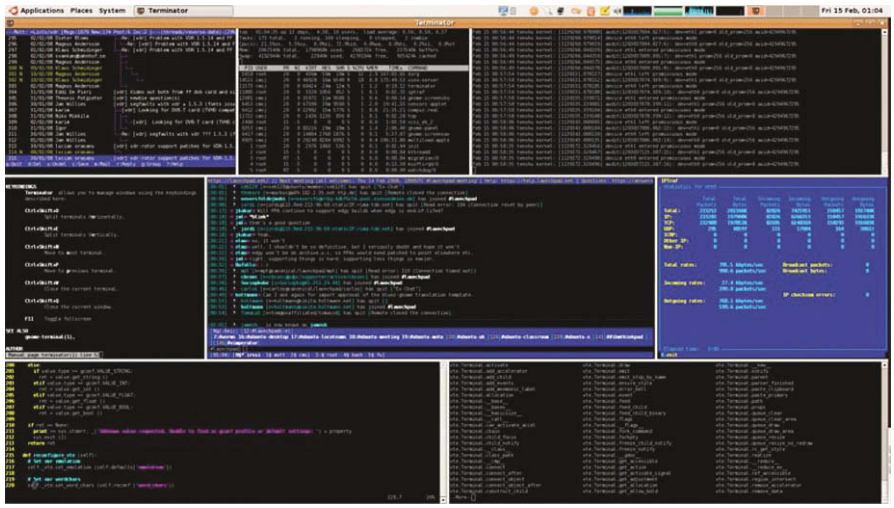
- **Meld :** Un outil pour gérer la fusion de fichiers. Il est capable de comparer simultanément 3 arborescences de fichiers. Son interface est très claire, ce qui dans ces phases, est essentiel.
- **iPython :** Un shell interactif pour Python.
- **ASCIIO :** Un éditeur de diagrammes qui a la particularité de faire des schémas en ASCII. Cet outil en perl est très pratique pour ajouter des schémas dans la documentation de ses sources.
- **Doxygen :** Un extracteur de documentations de codes source multi-langages. Il permet une bonne personnalisation, la génération des diagrammes de classes et autres fonctionnalités.
- **vim :** Un incontournable pour l'édition de fichiers. Il est capable de prendre en charge des fichiers de tailles importantes. Il demande un petit temps d'adaptation mais contient énormément de fonctionnalités.

D • Serveurs

Linux est utilisé sur près de 80% des serveurs. On retrouve donc tout naturellement beaucoup de solutions sous Linux. On retrouve donc pour les différents langages des serveurs adaptés et performants :



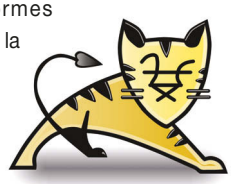
Eclipse



Terminator

Java/Java EE

- **Tomcat :** Un incontournable des conteneurs de servlets. Simple à configurer et administrer.
- **JBoss Wildfly 8 :** Le serveur d'application certifié JavaEE 7 le plus performant. Il est très modulaire et sa configuration se fait très facilement via ses fichiers de configuration. Aussi bien en production qu'en développement, Wildfly est très léger. Il prend en compte l'ensemble des normes JavaEE 7 et présente la meilleure intégration de ces différentes technologies.



- **Glassfish 4 :** Concurrent direct de Wildfly, Glassfish est une bonne solution pour des projets JavaEE. Il est cependant moins performant sur un environnement de développement et son intégration dans Eclipse n'est pas des plus idéales.

PHP

- **Apache :** C'est le serveur frontal de référence, robuste et complet. Il demande cependant une certaine maîtrise pour bien le configurer.
- **NGinx :** Une solution alternative à Apache. Il se montre plus simple dans sa configuration. Il contient également moins de modules, mais les principales fonctionnalités utiles dans Apache se retrouvent dans NGinx. Sa configuration est assez proche de celle d'Apache, la syntaxe y est homogénéisée et simplifiée.
- **Lighttpd :** Une solution plus simple qu'Apache et NGinx. Elle permet de mettre rapidement en place un frontal HTTP.

Ruby

- **Webrick :** Ce serveur embarqué permet d'exécuter le code Ruby en cours de développement.

- **FastCGI :** Les serveurs Apache, NGinx et lighttpd ont tous des modules FastCGI permettant l'intégration de Ruby. Cela demande cependant de bonnes connaissances en administration système. Idéal pour un environnement de production, cette solution se révèle un peu plus complexe sur un environnement de développement.
- **JRuby / Tomcat :** L'une des solutions pour avoir un serveur embarquant une application Ruby est de la compiler sur la JVM pour exécuter le code sur Tomcat. Cela permet de profiter de la puissance de la JVM de Java ainsi que des possibilités qu'offrent les serveurs tournant sur la JVM.

Python

- **Zope :** C'est la référence en termes de serveur d'application pour Python. Il se révèle tout aussi simple et performant en production qu'en environnement de développement.

CONCLUSION

Il n'existe pas de système parfait : cela dépend beaucoup des habitudes de travail et du domaine d'application.

Un bon développeur arrivera toujours à réaliser son code, quel que soit son environnement.

Linux offre un confort supplémentaire et une plus grande productivité pour ceux qui sont prêts à apprendre à utiliser le système pour automatiser leur travail.

Linux offre de grands avantages en termes d'optimisation de la puissance machine et des possibilités offertes par les commandes systèmes. Travailler sous Linux permet d'être d'avantage sensibilisé aux différentes étapes de fabrication d'une application.

On a une meilleure vision du développement, et ce jusqu'à la production.

Linux manque encore d'une vraie stratégie de communication et de publicité.

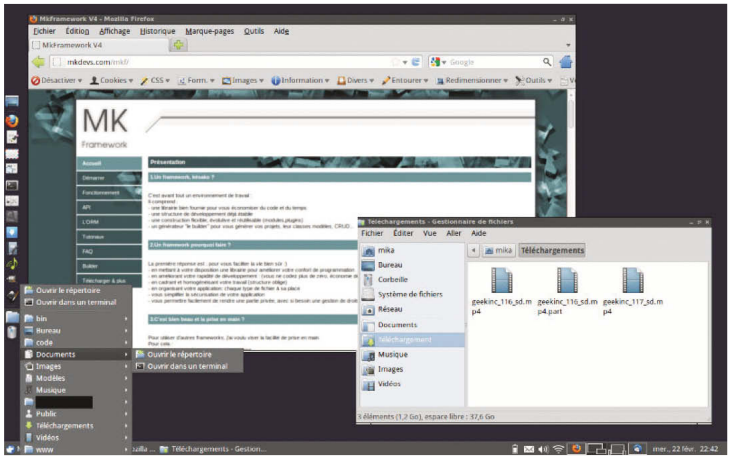
Cette étape est importante pour se faire connaître du grand public et inciter d'avantage de personnes à se poser la question «Et si on passait sous Linux ?»

Le système est gratuit et simple à installer : il ne tient qu'à vous de l'essayer !

Patrick **Guillerm**, *Architecte Java EE*

Patrick a débuté sa carrière en tant que graphiste 3D. Après avoir pris part à la réalisation de films d'animation grand public, il s'intéresse de plus près au développement et débute une formation intensive sur la technologie Java. Patrick couvre régulièrement les événements Palo IT. Il est également Leader de la Communauté Java chez Palo IT.

Témoignage



Vous développez pour Java, .Net, PHP, etc. sur un poste Linux ?

Je développe sous Linux en php et perl ! Pour php, j'utilise l'éditeur Geany, un terminal, et rapidSvn pour faciliter les merges, et autres différentiels svn. Pour simplifier la vie, je monte via sshfs les serveurs sur lesquels je travaille, ça permet de lancer un client lourd comme rapidSvn sur les fichiers du serveur ou faire des différentiels de fichiers avec Meld (outil équivalent à Kdiff sous Windows)

Les avantages de travailler sous Linux:

- Grep (bien pratique, combiné aux « pipe » appropriés pour trouver facilement des infos)
- multi-bureau: un par projet (avec le couple Chromium/Geany/Terminal/rapidSvn)
- les performances (si on utilise un gestionnaire de bureau léger bien sûr).

Comment choisir sa distribution ?

Cela dépend de son utilisation: si on privilégie les performances, je préconise xfce4 comme environnement de bureau et Debian/Ubuntu pour avoir une bonne compatibilité.

Après au choix: Xubuntu, Linux Mint Xfce, ou Voyager. Les avantages des distributions "basées" sur Ubuntu, c'est de bénéficier des efforts, accords ou simplement de l'appréciation d'autres projets. Dernier exemple en date avec Steam : Valve a proposé dès le départ son logiciel aux utilisateurs d'Ubuntu.

La logithèque peut permettre de faire le bon choix: en effet sous GNU/Linux on a 3 choix pour installer un logiciel

- il est disponible dans les dépôts officiels de sa distribution,
- il est disponible dans des dépôts non officiels ou PPA*: on les ajoute et on peut passer par son logiciel d'installation habituel (Ubuntu software, apt-get, aptitude),
- compiler les sources (configure/make/install).

Si on privilégie le confort, look and feel, on a le choix entre les bureaux KDE, Cinnamon, Gnome 3.

Ces environnements un peu plus gourmands en ressources, proposent une expérience qui se veut plus confortable en proposant notamment d'avoir une indexation au fil de l'eau de l'ensemble des fichiers pour les retrouver plus facilement. Ils proposent également un environnement plus « joli » avec des éléments en plus comme les widgets (post-it, liste de fichiers dans un répertoire, météo...).

KDE

Si je dois vous donner un conseil: testez-en plusieurs en "live DVD": en effet la grande majorité des distributions permet de voir en "démo" sans l'installer cela permet de voir le rendu , vérifier la bonne compatibilité avec votre matériel... D'ailleurs si vous ne savez pas quel environnement choisir, je vous invite à lancer le live de la distribution Hybryde Fusion http://www.hybryde.org/site/index.php qui permet de tester avec un seul DVD plusieurs environnements (sans l'installer).

Peut-on développer pour tout et n'importe quoi depuis un poste sous Linux ?

Personnellement je fais principalement des sites Web en php et des batchs perl.

A titre personnel, j'ai déjà développé en C++ / Qt et sur des applications Android.

On peut également faire des animations/applications Flash avec Haxe, on peut aussi faire du C# avec mono (et son ide monodevelop).

Les contraintes que l'on peut avoir en environnement GNU/Linux sont principalement le manque de certains logiciels (client Microsoft Sql Server, IIS..), on peut toujours les utiliser en installant une VM virtualbox.

Michael **Bertocchi**
Ingénieur développement
@dupot_org

Développer sa première application Windows Phone 8

Le développement d'applications Windows Phone est simple à appréhender pour ceux qui ont déjà une bonne connaissance de la plateforme .Net. C'est encore plus vrai si l'on possède des compétences dans l'utilisation des technologies WPF ou Silverlight. Néanmoins, il existe un certain nombre de spécificités liées à ce type de projet. Ce sont ces dernières que je vais essayer de vous faire découvrir au travers de cet article. Pour cela je vais vous donner les solutions aux problèmes que vous rencontrerez sûrement, comme moi lors de mon premier projet.

Prérequis

Avant de débuter un projet Windows Phone, il y a un certain nombre de prérequis qu'il faut satisfaire. Le premier et le plus évident est bien sûr d'avoir une installation de Visual Studio. Le second est d'obtenir le SDK Windows Phone 8. En effet son installation est indispensable pour créer un projet dans Visual Studio et pour disposer de l'ensemble des apis de la plateforme. Vous pouvez également mettre à jour l'émulateur si besoin. Sachez toutefois que depuis la version 2013 de Visual Studio, le SDK est inclus dans le package d'installation. Il n'est donc plus nécessaire de la télécharger séparément comme pour la version 2012. Cependant, il faut bien penser à le sélectionner lors de l'installation. Si cela n'a pas été le cas, il peut être ajouté en relançant l'installation.

Le dernier prérequis est de disposer d'une licence de développeur Windows Phone. Sans elle il sera tout de même possible de développer son application et de la tester avec l'émulateur. Par contre, elle deviendra indispensable pour publier son application sur le store ou encore, tester son application directement sur son téléphone. Il est également important de noter que désormais la licence permet aussi bien de développer une application Windows Phone que Windows Store. Le coût de cette licence est résumé ci-dessous : Fig.1.

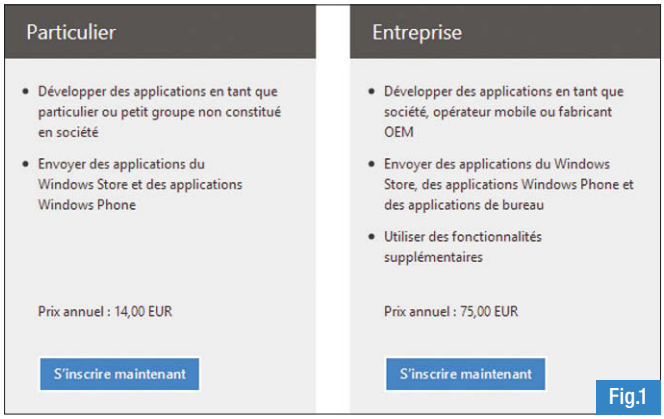


Fig.1

Présentation du projet de démonstration

Afin d'illustrer cet article et pour ne pas l'encombrer avec de trop nombreuses lignes de code, vous pouvez trouver sur Git Hub un projet qui illustre l'ensemble des points abordés. Pour cela, rendez-vous à l'adresse suivante : https://github.com/melcom/WindowsPhone-Meteo.

Ce projet contient une application qui fournit des informations sur la météo. Elle utilise l'ensemble des points abordés dans cet article et elle peut servir de socle technique pour débuter l'application de votre choix.

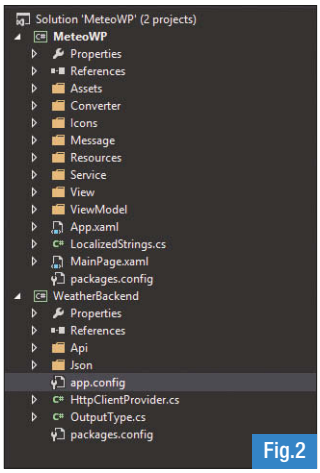


Fig.2

Architecture applicative

Comme pour une application WPF traditionnelle, je recommande l'utilisation du pattern MVVM pour structurer son application. Je conseille également d'écrire l'ensemble de son code métier dans une Portable Class Library et de l'exposer ensuite via une interface. Le principal avantage de cette méthode est, par exemple, de permettre la réutilisation de son code métier dans une application Windows Store. Sans oublier les avantages en termes de faible adhérence entre les composants et la testabilité.

Voici ce que cela donne avec l'application de démonstration : Fig.2.

On retrouve bien les deux projets : WeatherBackend et MeteoWP. Le premier contient la couche métier de l'application, celle qui est en charge de récupérer les informations sur la météo. Pour cela elle interroge un service REST distant et expose le résultat via une API. On isole ainsi parfaitement le fonctionnement interne de cette couche métier, que l'on peut ensuite facilement tester et réutiliser dans d'autres applications.

Le second projet contient quant à lui l'interface graphique. Comme évoqué précédemment, il utilise le pattern MVVM, on y retrouve donc des répertoires pour les vues, les ViewModels et les messages.

Navigation

Lorsque l'on débute une application Windows Phone, on s'attend à trouver des spécificités au niveau des composants utilisables ou bien encore à trouver des contraintes liées aux applications mobiles. On peut notamment citer la contrainte liée à la navigation entre les différentes vues. Dans une application WPF, pour ouvrir un nouvel écran, il suffit de l'instancier et de lui passer en paramètre les objets nécessaires à son fonctionnement. Dans le cadre d'une application Windows Phone le fonctionnement est totalement différent, il est plus proche de ce que l'on peut trouver dans une application Web. Pour naviguer d'une vue à une autre, il faut utiliser un service de navigation auquel on passera le chemin de la vue ainsi que la liste des paramètres sous forme de texte. Il n'est donc pas possible de passer par exemple pour une vue de détail un objet déjà initialisé. De plus, ce service de Navigation n'est disponible que dans le code des vues, impossible donc d'y accéder depuis un ViewModel, ce qui est très pénalisant dans notre cas. Il faut donc trouver un moyen d'y accéder car les changements de navigation sont toujours à l'initiative des ViewModels. Pour résoudre ce problème, la première chose à faire est de déclarer dans le code de la vue principale un message qui appellera le service de navigation comme le montre l'exemple ci-dessous :

```
Messenger.Default.Register<GlobalNavigationMessage>(this,
message => NavigationService.Navigate(message.ViewUri));
```

La classe GlobalNavigationMessage permet d'obtenir à partir du nom de la vue son Uri. Ainsi, lorsqu'un message contenant une instance de cette classe sera envoyé, le service de navigation sera appelé avec l'adresse de la vue à charger. La classe GlobalNavigationMessage prend également en charge la transmission des paramètres. Je ne détaillerai pas ici son implémentation, je vous laisse la consulter dans le projet de démonstration. Il reste un dernier point à découvrir : comment récupérer les paramètres dans la vue qui vient de s'ouvrir. Cette opération s'effectue en deux étapes. La première consiste à surcharger la méthode OnNavigatedTo dans la classe de base des vues. Ainsi, comme on peut le voir dans le morceau de code suivant, il est possible de récupérer à partir de l'objet QueryString l'ensemble des paramètres.

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
 base.OnNavigatedTo(e);

 var viewModel = DataContext as NavigationViewModelBase;
 if (viewModel != null)
 {
   viewModel.SetProgressVisibility(a => progressIndicator.
IsVisible = a);
   viewModel.AddParameters(NavigationContext.QueryString);
 }
}
```

L'objet QueryString est un dictionnaire qui contient pour chaque paramètre un couple clé/valeur. C'est ce dictionnaire qui est ensuite transmis au ViewModel via la méthode AddParameters qui est définie dans le ViewModel de base. Les différents paramètres seront alors accessibles depuis le ViewModel associé à la vue.

Composants graphiques

La plateforme Windows Phone comporte un certain nombre de contrôles graphiques. On retrouve bien évidement l'ensemble des composants standards (bouton, texte, image...), mais quelques-uns sont spécifiques. C'est par exemple le cas pour le LongListSelector qui permet d'afficher toutes sortes d'informations sous forme de liste. Ce composant est probablement l'un des plus complexes car il permet de répondre à un grand nombre de cas de figures. Cependant le composant sur lequel on doit porter une attention particulière est l'ApplicationBar, composant permettant d'afficher en bas d'écran un ensemble de boutons. Celui-ci ne supporte pas le databinding. Du coup, il n'est pas possible de lier un bouton avec le ViewModel pour effectuer une action. Le libellé du bouton ne peut pas non plus être défini dans le xaml avec la méthode de régionalisation que l'on verra un peu plus loin. Malgré toutes ces limitations, il est tout de même possible d'utiliser ce composant dans une application MVVM. La première chose à faire est de décrire l'ApplicationBar dans le constructeur de la classe de la vue comme le montre l'exemple suivant :

```
var settings = new ApplicationBarIconButton(new Uri(«/Icons
/appbar.settings.png», UriKind.Relative));
settings.Text = AppResources.Settings;
settings.Click += SettingsClick;
ApplicationBar.Buttons.Add(settings);
```

Il faut ensuite gérer le clic sur le bouton. Une solution consiste à utiliser un message qui aura pour objectif de propager l'action associée au bouton vers le ViewModel. Ou bien, comme le montre l'exemple ci-dessous, ouvrir une vue directement.

```
private void SettingsClick(object sender, EventArgs e)
{
Messenger.Default.Send(new GlobalNavigationMessage(ViewList.
Settings));
}
```

Il reste un dernier point auquel il faut faire attention lors de l'utilisation d'une ApplicationBar : lorsque l'on clique sur un des boutons, le contrôle qui porte le focus ne le perd pas, et le databinding ne se met pas à jour. C'est notamment le cas dans l'écran de configuration de l'application. Il faut donc forcer la mise à jour avant de pouvoir récupérer la valeur du dernier champ texte édité. Voici un exemple de méthode qui effectue cette tâche :

```
protected void UpdateBinding()
{
 var currenTextBox = FocusManager.GetFocusedElement() as TextBox;
 if (currenTextBox != null)
 {
   BindingExpression binding = currenTextBox.GetBindingExpression
(TextBox.TextProperty);
   if (binding != null)
     binding.UpdateSource();
 }
}
```

Gestion de la résolution

Avec Windows Phone 7.1, tous les écrans de téléphone avaient la même résolution WVGA. Mais avec l'arrivée de la version 8, trois nouvelles résolutions sont prises en charge :

Nom résolution	Résolution	Format	Echelle	Résolution mise à l'échelle
WVGA	480 800	15:9	1x	480 800
WXGA	768 1280	15:9	1.6x	480 800
720p	720 1280	16:9	1.5x	480 853
1080p	1080 x 1920	16:9	1.5x	480 853

Pour gérer facilement l'ensemble des résolutions disponibles, les applications n'ont accès qu'à une résolution mise à l'échelle et non pas à la résolution physique. De ce fait le développeur utilise quasiment toujours la même résolution quel que soit le téléphone utilisé. La seule différence réside dans le format de l'écran, qui ajoute 53 pixels en longueur dans le cas d'un écran 16 :9. Cependant, bien que la résolution logique soit quasiment toujours identique, il est par exemple préférable de ne pas fixer la taille des composants, ou encore les largeurs de colonnes dans les grilles. Pour cela il faut utiliser la définition de taille avec la notation en * ou avec Auto comme dans l'exemple ci-dessous :

```
<Grid.RowDefinitions>
  <RowDefinition Height=»1*» />
  <RowDefinition Height=»3*» />
  <RowDefinition Height=»2*» />
</Grid.RowDefinitions>

<Grid.RowDefinitions>
  <RowDefinition Height=»Auto» />
</Grid.RowDefinitions>
```

Paramétrage

Le paramétrage est un élément important d'une application, et dans le cadre de la plateforme Windows Phone, il est relativement facile à mettre

en place. Pour ce faire l'api Windows Phone fournit une sorte de dictionnaire de données appelé ApplicationSettings que l'on peut appeler via l'objet IsolatedStorageSettings. Pour l'utiliser, rien de plus simple, il suffit d'ajouter un élément dans le dictionnaire puis d'appeler la méthode Save comme dans l'exemple ci-dessous :

```
IsolatedStorageSettings settings = IsolatedStorageSettings.
ApplicationSettings;
settings[«City»] = «Paris»;
settings.Save();
```

Régionalisation

La régionalisation n'est pas la première chose à laquelle on pense lors du développement d'une application. Cependant dans le cas d'une application Windows Phone partagée sur le Store, cette étape me semble incontournable. En effet, l'objectif est d'avoir un maximum de visibilité et de faire en sorte qu'un maximum de personnes soient intéressées par l'application. Or plus on multipliera les langues supportées, plus la portée du projet sera grande. Il y a également un autre avantage à utiliser cette technique : regrouper l'ensemble des textes. Cela permettra par exemple d'éviter d'avoir un bouton « sauvegarder » dans une vue et un bouton « enregistrer » dans une autre. Un autre point important à signaler est que la mise en place des différentes langues est très rapide à réaliser, la première étape consistant à sélectionner les langues souhaitées. Pour ce faire, il faut se diriger vers les propriétés du projet puis dans l'onglet « Application » pour sélectionner les nouvelles langues : Fig.3.

Par défaut seul l'anglais est supporté et chaque nouvelle sélection ajoutera dans le répertoire « Resources » un nouveau fichier resx. C'est ce fichier qu'il conviendra de remplir par la suite en faisant bien attention d'avoir les mêmes champs dans tous les fichiers.

Pour terminer avec ce chapitre voyons comment utiliser ces différentes ressources. Là encore c'est très rapide, la première ligne permet d'utiliser la ressource traduite dans le xaml, tandis que la deuxième ligne permet de faire la même chose dans le code C# :

```
Text=»{Binding Path=LocalizedResources.ApplicationTitle, Source=
{StaticResource LocalizedStrings}}»

refresh.Text = AppResources.Refresh;
```

Débogage & tests

Après quelques heures de développement viendra le temps de tester une première version de votre application. Pour cela, la solution la plus simple est d'utiliser l'émulateur Windows Phone. Il vous suffit d'appuyer sur F5 et l'application se chargera, et sera testable immédiatement. Il y a tout de même quelques limitations dues au fonctionnement même de l'émulateur : ce dernier utilise la technologie Hyper-V pour héberger le système d'exploitation mobile de Microsoft. Il faut donc une machine capable de faire tourner ce système de virtualisation. Une autre fonctionnalité très intéressante de l'émulateur et de Visual Studio est de pouvoir simuler la qualité du réseau auquel on se connecte. Ce point est important car, lorsque l'on développe et teste une application sur sa machine, l'OS mobile virtualisé utilisera votre connexion réseau et aura donc un accès très rapide à Internet. Or on le sait tous, avec un smartphone, les connections cellulaires ont un débit qui varie énormément en fonction de la couverture réseau. Il faut donc tester ces cas de figures. Pour cela il suffit d'aller dans le menu « Tools » puis dans la rubrique « Simulation Dashboard » pour avoir accès à l'écran ci-dessous : Fig.4.

On peut alors limiter la vitesse de la connexion ainsi que simuler une baisse de qualité du réseau. Un autre avantage intéressant de l'émulateur est la mise à disposition d'environnements de différentes résolutions. Vous aurez donc la possibilité de vérifier comme évoqué précédemment que l'application s'adapte parfaitement aux nombreuses résolutions supportées par Windows Phone 8.

L'émulateur permet donc d'avoir un environnement complet de tests, mais il ne remplacera pas totalement un téléphone physique. Ce dernier est même indispensable dans au moins un cas de figure : le multitouch. En effet, le simulateur utilise la souris pour émuler l'action du doigt sur l'écran. Or il n'y a qu'un seul pointeur de souris sur une machine. Impossible donc de tester les gestes qui nécessitent plusieurs points de contacts avec l'écran. Mais avant de pouvoir utiliser un téléphone comme support physique, il faut le débloquer pour pouvoir y installer l'application en cours de développement. Pour cela Microsoft fournit un outil nommé « Windows Phone Developer Registration » comme le montre la capture ci-dessous : Fig.5.

Il suffit ensuite de suivre les quelques étapes de l'outil pour déverrouiller son téléphone. Puis, en sélectionnant « device » à la place de l'émulateur et en laissant son téléphone branché, il est possible d'y installer l'application et de la déboguer. Il faut néanmoins veiller, lors de la phase de déploiement, à ce que le téléphone soit bien déverrouillé. Sinon cette étape échouera.

Conclusion

J'ai essayé de vous faire découvrir au travers de cet article les quelques éléments qui me semblent nécessaires pour bien débuter le développement d'une application mobile pour Windows Phone. De plus, l'application de démonstration contient l'ensemble des éléments présentés de manière concrète. Vous devriez désormais avoir toutes les cartes en main pour développer une nouvelle application et enrichir le Store de Microsoft.

Pierre-Henri **Gache**
Consultant chez Cellenza
- Software Development Done Right
Son Blog: http://pierrehenrigache.azurewebsites.net

Cellenza

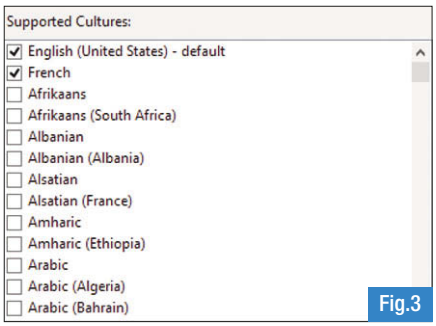


Fig.3

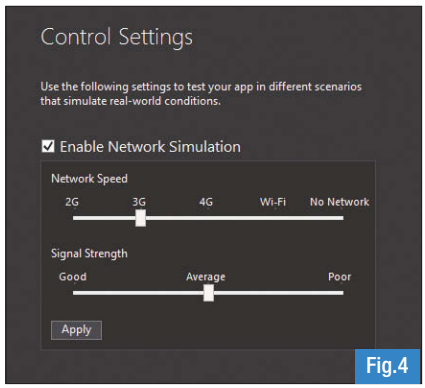


Fig.4

Fig.5

Comment déclarer les revenus d'une App ?

Si comme moi vous avez votre entreprise (SARL, SA, EURL ...) il est indispensable de déclarer vos revenus d'applications Windows Store / Windows Phone. Bien que ce ne soit pas nouveau, je n'ai pas trouvé d'explications claires. J'ai donc mené ma propre enquête et consulté plusieurs comptables. Mais notez que ceux qui n'ont pas su me répondre sont les agents du fisc !

SPÉCIAL IMPÔT

Dans les faits

Vous avez des applications qui sont vendues par Microsoft Corp. Cette dernière vous reverse une somme dès qu'un seuil est atteint (et surtout dès qu'elle a reçu les fonds des différents opérateurs mondiaux). Microsoft vous reverse 70%. Le mécanisme est quelque peu complexe car selon les différents taux de change, entre le moment où l'utilisateur achète votre application ou produit (in-app purchase), et le moment où Microsoft vous reverse le montant, il y a quelques jolies virgules qui, au final, ne donneront pas un chiffre tout rond.

Que faire pour être en règle ?

Dès que vous avez un virement sur votre compte pour Windows Phone, rendez-vous sur votre compte DevCenter https://dev.windowsphone.com où il va falloir éditer un document disponible à la rubrique REPORT / FINANCIAL SUMMARY : Fig.1.

Un fichier Excel va être généré. Afin de vous y retrouver plus tard, je vous invite à y ajouter une colonne et à faire ajouter une « somme » au bas de celle-ci. Puis vous imprimerez ce tableau car votre comptable sera content (non pas comptant hein!) Fig.2.

Pour les applications Windows Store https://appdev.microsoft.com, dans la rubrique « Tableau de bord / Récapitulatif financier », vous retrouverez une rubrique « Historique des paiements », où on peut afficher, puis imprimer le détail qu'il nous faut. Fig.3.

Ensuite il faut facturer Microsoft ! Et c'est tout à fait logique que vous éditiez une pièce car à défaut, le FISC pourra vous demander 20% de TVA ! C'est tout !

En gros vous allez facturer à Microsoft une prestation de services immatérielles; ça ne sert à rien de leur envoyer une copie de la facture, ils la mettront immédiatement à la poubelle!

Pour notre tranquillité et pour que notre dossier soit complet, on ajoutera tout de même cette mention correspondant à l'exonération de TVA *« Exonération de TVA, art. 259-1 du Code Général des Impôts. Services soumis au mécanisme d'autoliquidation – art. 283-2 du CGI »*

L'adresse de facturation est :

Microsoft Luxembourg, SARL
23-29, Rives de Clausen
L-2165 Luxembourg

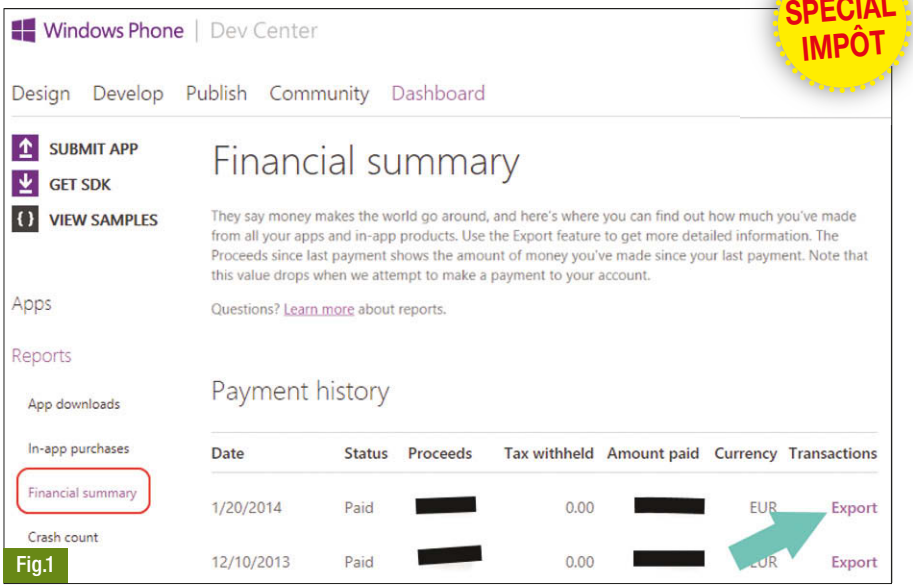


Fig.1

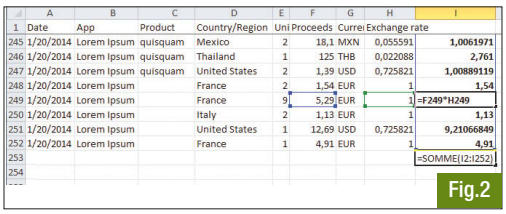


Fig.2

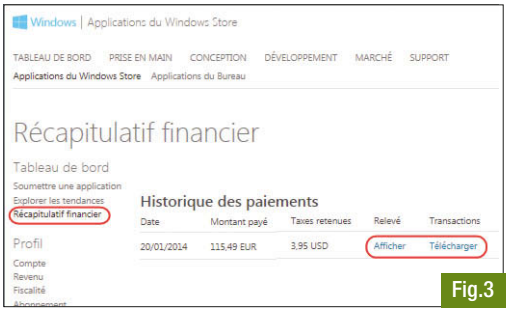


Fig.3

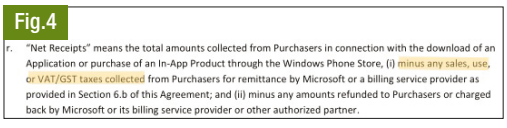


Fig.4

Fig.5

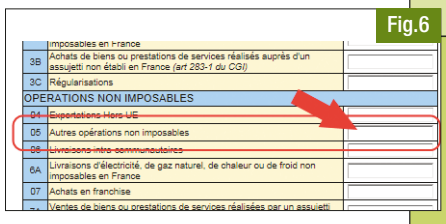


Fig.6

Mais qui paie la TVA alors ?

C'est Microsoft qui la reverse. Et c'est noté dans le contrat : » WINDOWS PHONE STORE APPLICATION PROVIDER AGREEMENT » Fig.4.

Dois je faire une DES?

Une DES est une « Déclaration européenne de services ». Bien que le libellé sur notre compte bancaire est ***Microsoft Corp***, le paiement est réalisé via sa filiale européenne au Luxembourg qui possède son numéro de TVA intracommunautaire : **LU19878750**. Une DES est donc obligatoire. Rendez-vous donc sur le site https://pro.douane.gouv.fr/ avant le 11 de chaque mois pour la déclaration mensuelle et ajoutez-y vos revenus d'apps. Fig.5.

Et c'est tout?

Non, il va falloir aussi, mensuellement, si toutefois vous êtes avec la TVA acquittée à l'encaissement, déclarer vos montants reçus avec votre déclaration de TVA à la ligne 05 « Autres opérations non imposables » : Fig.6.

Christophe **Peugnet**
Gérant et développeur des logiciels SodeaSoft de la société EBLM
http://www.sodeasoft.com
Blog : http://www.peug.net - Twitter : @tossnet1

Le développeur, l'avenir de la France ?

Le 6 mars dernier, un rapport a été publié par le ministère de l'innovation et de l'économie numérique : « les développeurs, un atout pour la France ». Nous avons voulu revenir dessus. Car si ce rapport est louable sur plusieurs éléments, il nous paraît éloigné des problématiques sur d'autres.

Le constat du rapport : la quasi-totalité des grands acteurs du numériques sont américains. Malgré les talents, la qualité de la formation, la France ne conquiert pas ce « nouveau monde », à l'exception de quelques dizaines d'éditeurs, laboratoires et start-ups.

Un des problèmes sera le manque de confiance dans les développeurs et les entrepreneurs. La prise de risque est souvent trop limitée pour inciter les initiatives disruptives. Les aides, les financements sont des points importants, mais sans marché, ces entreprises n'ont pas d'avenir en France, ou ailleurs. Le rapport met en avant l'open source, les API des administrations publiques. Et surtout, favoriser l'accès aux marchés publics de ces entreprises innovantes. Pour le rapport, la France s'est beaucoup concentrée sur les grandes entreprises, les grandes filières industrielles. Celles-ci ont capté, et captent toujours, une grande partie des budgets et des commandes publiques. Mais la France est absente des systèmes, des navigateurs et des outils de développements (il faut un poil nuancé notamment avec les langages de programmation).

Les recommandations du rapport

Le rapport évoque plusieurs pistes :

1 Prendre en compte le rôle essentiel du développeur : le développeur est encore trop souvent considéré comme un exécutant. Et c'est parfois un objet codant mal identifié,
2 Une feuille de route technologique pour l'Etat, les ministères, les opérateurs publics,
3 Une feuille de route du gouvernement gagnerait à être complétée par une feuille de route technologique notamment sur la mobilité, les standards web, les API...
4 Avoir un Github français : les services de l'Etat, les collectivités, pourraient déposer et accéder aux codes, aux API. Ceci serait accessible aux développeurs,
5 Promouvoir les développeurs dans l'administration : les développeurs à des postes de responsabilités pour conduire les projets numériques,
6 Adapter les conditions d'investissement pour soutenir les projets technologiques : le financement classique n'est pas adapté aux start-ups disruptives,
7 Formation des dévelo0ppeurs : les recruteurs ont parfois du mal à trouver les bonnes compétences ; il y a pénurie sur des postes de haut niveau. Continuer à former des ingénieurs, mais aussi à bac+2. Détecter dès l'école primaire les talents en éveillant les élèves à la programmation. « On pourrait aussi mettre en place, en ciblant de manière prioritaire les banlieues, des «écoles du numérique» destinées à des jeunes de 18 à 25 ans «décrocheurs». Ces dispositifs peuvent s'inspirer de réalisations existantes comme "Web@cademie", «42» ou «codeacademy.org»... » indique aussi le rapport,
8 Visa de travail pour les développeurs venant en France,
9 Ces propositions réactivent la question du CTO, un responsable au plus haut niveau de l'Etat, pour coordonner la plate-forme technologique France et valoriser le pool de technologies et de codes développés en France.

Commentaire de la rédaction

Le développeur mal aimé de l'informatique mais mal nécessaire ? Depuis 20 ans nous entendons peu ou prou la même chose. Plusieurs éléments sont peu ou pas abordés : la formation, le revalorisation des métiers du développement, les salaires.

1 la formation

Pour remédier à la pénurie de talents (le terme pénurie est à manipuler avec prudence car le chômage existe dans ce secteur), il faut des cycles courts type bac+2, parallèlement à la formation de niveau ingénieur. Former un développeur n'est pas un problème en soi, former un bon développeur, là c'est autre chose.

« 42 » et l'alternance sont deux pistes intéressantes mais il faut tout de même que le candidat ait un minimum de compétences à la base. On ne s'improvise pas développeur. Souvenez-vous des années 2000 quand tout le monde, ou presque, se prétendait développeur web, webmaster. Le résultat fut parfois catastrophique.

Faut-il la quantité à la qualité ? Il faut clairement former plus de développeurs. Oui, il faut faire découvrir le développement, la programmation le plus tôt possible. Mais ensuite, il faut aussi la motivation du jeune, de l'étudiant. Ce n'est pas parce que l'enfant a un parent développeur ou passionné de technologie qu'il va obligatoirement faire de l'informatique. Ensuite, il faut aussi revaloriser le développement, le développeur, dans les cursus existants. Il faut donner envie aux étudiants de le devenir et de le rester. Trop souvent, on entend encore demain je veux être chef de projet, architecte, etc. D'autre part, la culture du diplôme est encore trop importante en France. On n'embauche pas sur la compétence mais souvent sur le diplôme du candidat. Le mot d'ordre doit être : faire carrière dans le développement, oui c'est possible en France.

2 Faire une carrière de développeur

A Programmez, nous connaissons de nombreux développeurs qui font du développeur leur métier et espèrent y rester. Malheureusement cette « positive attitude » est loin d'être partagée par une majorité des développeurs. Le métier de développeur est encore mal perçu. « Tu ne vas pas faire du code toute ta vie » ! Il y a un problème sociétal à résoudre. Le « pisseur de code » a la vie dure. En France, les grandes entreprises et les SSII sont les premiers employeurs de développeurs. Le développeur y sera souvent anonyme, noyé dans la masse. Le turn-over y est très élevé. Certaines sociétés voient le développeur comme un simple ouvrier spécialisé, sans valeur ajoutée. Heureusement, certains secteurs revalorisent le développement et offrent un cadre de travail motivant, avec parfois, des heures libres pour des projets personnels, la veille technologique, la formation.

3 Le salaire

Pour nous, le salaire est un élément important. En France, toutes les études montrent que le développeur démarre à environ 27 – 32 k€ par an (parfois -25 k€ pour un développeur web). Les profils experts, les compétences les plus rares peuvent atteindre 45 – 55 k€, à de rares exceptions, +60 k€. Dans le jeux vidéo, les salaires peuvent être plus bas, les candidats sont nombreux et prêts à travailler sans compter. Si nous prenons l'exemple des Etats-Unis, le développeur peut y faire carrière, être un salarié crucial d'une entreprise, et devenir une véritable star dans son domaine. Les salaires moyens sont en conséquences : un développeur mobile expert peut espérer gagner entre 100 – 144 000 $, un ingénieur logiciel sera entre 89 et 137 000 $, un développeur entre 69 et 122 000 $. (chiffres Forbes).

Venez en discuter sur
http://programmez.com/barcamp

Rapport : http://www.redressement-productif.gouv.fr/files/20140306_rapport_tariq_krim.pdf

A la Une du numéro de mai de L'Informaticien.

La réflexivité en Objective-C

Depuis plus de 40 ans, les langages n'ont cessé d'évoluer. Du binaire à l'hexadécimal, de l'assembleur au langage compilé, des langages interprétés aux langages avec du code managé, les évolutions continueront tant que l'informatique sera d'une importance capitale.

Mais qu'est-ce que la réflexivité ?

Il s'agit d'une capacité qu'un langage informatique peut avoir, lui permettant de pouvoir consulter et modifier ses structures internes au sein même du programme et lors de son exécution.

EN DÉTAIL

Pour résumer, le développeur dans le cadre d'un langage objet met en place un modèle conceptuel, basé principalement sur des classes lui permettant de se représenter plus facilement un système réel. Cependant, cette représentation n'a de sens que pour le développeur, lorsque le langage sera compilé, celui-ci aura été transformé en langage machine et n'existera plus sous la forme d'un modèle objet. Cependant, l'apparition des langages semi-compilés, basés sur du « code managé », a permis l'élaboration du concept de réflexivité.

Comme le langage n'est plus directement compilé en langage processeur mais en byte-code, qui sera compilé (ou interprété) à la volée par la machine virtuelle, celui-ci peut contenir des métadonnées. Cela permet d'avoir des fonctionnalités très agréables pour les développeurs : stacktrace, réflexivité, etc. Ainsi la machine virtuelle a bien connaissance du modèle objet, et permet cette retranscription à l'exécution tout en faisant abstraction de cela lorsqu'elle envoie le code au processeur. Les systèmes réflexifs se scindent en deux catégories :

- l'introspection.
- l'intercession.

Le premier permet au langage, et donc au développeur, d'examiner la structure de l'application à un moment donné. Cependant, celui-ci ne peut être modifié. La seconde, l'intercession, est bien plus complexe, car en plus de pouvoir examiner la structure de l'application, le langage fournit les outils pour pouvoir la modifier au bon vouloir du développeur durant son exécution. Comme expliqué plus haut, les machines virtuelles (runtime) permettent d'avoir une connaissance du modèle objet. Cela se traduit par l'apparition des méta-classes. Ces objets ne sont pas des classes définies par le développeur. Elles sont utilisées par l'environnement d'exécution pour le débuggage, les stacktraces et la réflexion.

De nombreux langages supportent le concept de réflexion:

- SmallTalk ;
- Ruby ;
- Python ;
- Java ;
- Objective-C.

Les langages utilisés par Microsoft sont un cas particulier. A l'inverse de la philosophie Java, Microsoft a développé une architecture complète de code managé : .NET. Ainsi, tous les langages utilisés par Microsoft sur leur plateforme de développement basée sur .NET, ont une capacité réflexive: C#, Visual Basic, C++/CLI, etc.

Un exemple en Java

Prenons une classe Voiture possédant une méthode démarre.

Voici comment nous pourrions utiliser cette classe avec la réflexion :

```
Class voitureClass = Class.forName("Voiture"); Object aVoiture
 = voitureClass.newInstance();  Method demarreMethod = voitureClass.
getMethod("hello", null); demarreMethod.invoke(aVoiture, null);
```

OBJECTIVE-C

Le langage est défini comme un **langage de programmation orienté objet réflexif**. Pour rappel, l'Objective-C est une extension du C, qui, pour sa part, n'est ni orienté objet, ni réflexif.

Comme nous l'avons vu précédemment, la réflexivité peut s'effectuer si le code compilé n'est pas exécuté directement par le processeur mais interprété par un processus distinct se chargeant d'envoyer les instructions au processeur.

L'Objective-C fonctionne dans un environnement d'exécution (runtime) qui a pour particularités d'être écrit en C, d'être très léger et embarqué dans l'application elle-même.

Ainsi, ce processus analyse le code Objective-C et l'exécute au moment donné, permettant donc d'avoir une réflexion sur le code qu'il exécute.

Manipuler le runtime Objective-C

La manipulation du runtime en Objective-C passe par l'utilisation de méthodes écrites en C, que l'on peut retrouver dans les fichiers d'en-têtes suivants :

- MacTypes.h
- NSObjCRuntime.h
- objc/message.h
- objc/objc-api.h
- objc/objc-auto.h
- objc/objc.h
- objc/runtime.h

(Ces fichiers ne sont pas tous nécessaires pour les exemples que nous allons voir ensuite.)

Les classes

La plupart des méthodes nécessaires à la manipulation des classes sont préfixées par class_.

Prenons par exemple, une méthode permettant d'obtenir la superclass d'une classe :

```
Class aSuperClass = class_getSuperclass(aClass);
```

Pour les objets primitifs du Framework Foundation, vous obtiendrez la classe NSObject, qui est la classe principale de tous les objets de ce framework. Même si la majorité des fonctions sont préfixées par class_, certaines, très utiles, ne le sont pas.

Par exemple, la méthode permettant de créer une nouvelle classe :

```
Class theNewClass = objc_allocateClassPair(superClass, "New
ClassName", 0);
```

Cette fonction vous permettra de créer votre modèle objet en ligne de code par exemple. Voyons en détail quelques méthodes.

class_addIvar

Cette méthode permet d'ajouter un Ivar à une classe. Pour rappel, les Ivar représentent des variables liées à une instance de classe. Ces variables sont propres à une instance, mais ne sont pas accessibles depuis l'extérieur de la classe (vs Property)

class_addMethod
Cette fonction permet d'ajouter une méthode à une classe. L'ajout d'une méthode se fait avec trois paramètres : un selector définissant le nom de la méthode, un pointeur vers une méthode C se chargeant de l'implémentation de cette nouvelle méthode et, pour finir, les types correspondants à la nouvelle méthode (type de retour et paramètre).

class_addProtocol
Cette fonction permet d'ajouter un protocole à une classe.

class_createInstance
Celle-ci permet de créer une instance de classe. Cette méthode est notamment utilisée par la classe NSObject lorsque qu'on fait un alloc/init sur une classe.

class_getClassMethod
Cette fonction permet d'obtenir une méthode de classe (+ ou méthode statique). L'information retournée est de type Method, sous forme de structure représentant une méthode de classe.

class_getClassVariable
Cette fonction permet d'obtenir l'Ivar d'une classe. L'objet retourné est de type Ivar qui est une structure permettant d'obtenir des informations sur la variable elle-même.

class_getInstanceMethod
Cette fonction permet d'obtenir une méthode d'instance (-).

class_getInstanceVariable
Celle-ci permet d'obtenir l'Ivar d'une instance de classe.

class_getName
Cette méthode permet d'obtenir le nom d'une classe à partir de l'objet Class la désignant.

class_getSuperclass
Cette fonction permet d'obtenir la Class représentant la superclass d'une classe.

class_getProperty
Cette fonction permet d'obtenir une structure de type objc_property_t permettant de représenter une propriété de classe.

class_getVersion
Cette fonction permet d'obtenir la version de la classe. Il est également possible d'utiliser la méthode class_setVersion pour définir le numéro de version d'une classe.
Cette fonctionnalité peut s'avérer très utile pour la sérialisation/désérialisation d'un modèle objet à partir de métadonnées (XML, JSON, etc.)

Les protocoles
Les protocoles sont les pendants des interfaces Java en Objective-C. Ce sont des définitions de méthodes qu'une classe peut adopter. Cela permet d'avoir un comportement type avec toutes les classes se conformant à un protocole particulier.
Ils sont notamment utilisés pour mettre en place le design pattern delegate, très utilisé en Objective-C.
Nous avons vu précédemment qu'une méthode existait dans le runtime pour ajouter un protocole à une classe.
Il existe aussi un certain nombre de méthodes permettant de manipuler un protocole. Ces méthodes sont pratiquement toutes définies avec le préfixe protocol_. Par exemple :

```
char* name = protocol_getName(aProtocol);
```

Retourne le nom du protocole défini par la structure de type Protocol passé en paramètre.

Comme pour les classes, il existe quelques méthodes qui ne sont pas préfixées par protocol_ mais par objc_. Ces dernières permettent de créer ou de dupliquer un protocole.
Observons les méthodes les plus essentielles en détails :

protocol_getName
Comme pour les classes, cette fonction retourne le nom du protocole.

protocol_addProperty
Cette méthode ajoute une propriété à un protocole (@property). Contrairement aux classes, l'ajout d'une propriété dans un protocole ne crée pas automatiquement l'Ivar associé (voir les "Associated Object").

protocol_addMethodDescription
Cette fonction ajoute une méthode au protocole. On parle ici de "description" puisque les protocoles ne fournissent pas d'implémentation.

protocol_copyMethodDescriptionList
Cette fonction retourne une liste de description de méthode dudit protocole.

protocol_getProperty
Cette fonction retourne une propriété.

protocol_copyPropertyList
Cette fonction retourne une liste de propriétés du protocole.

Les méthodes et les propriétés
Nous avons précédemment vu quelques méthodes pour accéder aux propriétés et aux méthodes d'une classe (ou d'un protocole).
Les informations retournées par ces méthodes sont principalement des structures. Certaines sont directement utilisables. D'autres nécessitent l'utilisation de certaines méthodes du runtime pour obtenir des informations supplémentaires.
Les fonctions liées à la manipulation des méthodes sont préfixées par method_ et celles pour les propriétés sont préfixées par property_.
Observons ensemble les principales méthodes :

method_invoke
Permet d'invoquer l'appel d'une méthode sur une instance d'objet particulière.

method_getName
Cette fonction retourne le nom de la méthode sous forme de SEL (selector).

method_getImplementation
Cette méthode retourne l'implémentation d'une méthode.

method_getReturnType
Celle-ci retourne le type de retour.
Le type de retour est indiqué dans un format normalisé disponible à cette adresse : http://goo.gl/IfgXUC

method_getNumberOfArguments
Cette fonction retourne le nombre d'arguments à la méthode.

method_getArgumentType
Cette fonction retourne le type d'un argument précis dans la méthode passée en paramètre.

method_setImplementation
Permet de définir l'implémentation d'une méthode (via un pointeur de fonction).

method_exchangeImplementations
Cette fonction, un peu particulière, permet d'échanger l'implémentation de deux méthodes.

property_getName
Retourne le nom d'une propriété.

property_getAttributes
Cette fonction retourne les attributs associés à une propriété. Les attributs sont formatés dans une chaîne de caractères. Vous trouverez plus de détails sur le format ici : http://goo.gl/l6mu5h

Les messages
Une classe en Objective-C n'est qu'un conteneur virtuel de structures et de définitions de méthodes.

Lorsque nous invoquons l'appel d'une méthode en Objective-C, cela se fait par le mécanisme de message. Ces messages sont transparents pour le développeur et ressemblent à l'appel d'une méthode.

Ce système de messages est géré par le runtime en Objective-C.
Des méthodes existent pour faire de l'appel de messages avec des fonctions C, sans passer par les classes elles-mêmes :

objc_msgSend
Cette méthode permet d'envoyer un message à un objet. Le prototype de cette méthode est le suivant :

```
id objc_msgSend(id self, SEL op, ...)
```

Autrement dit, pour envoyer un message à un objet, il faut utiliser la fonction comme tel :

```
id returnValue = objc_msgSend(theObject, @selector(aGood
Method));
```

Ici il s'agit d'une méthode sans arguments. La méthode objc_msgSend est à argument variable. Cela permettra notamment de définir les arguments passés à la méthode pointée par le message. Par exemple :

```
id returnValue = objc_msgSend(theObject, @selector(hello:
howAreYou:whereDoYouLive:), @"Vincent", @"Fine", @"Macon");
```

objc_msgSendSuper
La méthode objc_msgSendSuper est une méthode permettant non pas de cibler l'objet passé en paramètre mais sa classe mère.

Cette méthode est utilisée par le runtime lorsque le développeur effectue un envoi de message super : [super helloWolrd];

Autres fonctionnalités
Le runtime et les méthodes disponibles représentent la base utilisée par le langage pour fonctionner. La majorité des fonctionnalités d'Objective-C sont donc disponibles.

sel_getName
Cette fonction permet de retourner le nom d'un SEL (selector) sous la forme d'une chaîne de caractères.
A ne pas confondre avec la fonction getName sur une méthode qui retourne celui-ci sous forme de selector.

sel_registerName
Cette fonction est très importante, elle permet d'enregistrer un nom de méthode à un selector dans le runtime.
Afin d'être capable de faire les associations, le runtime doit maintenir un registre des méthodes et selector disponibles. Cette étape est nécessaire avant d'ajouter une méthode à une classe.

sel_isEqual
Permet de vérifier l'égalité entre deux selector.

imp_getBlock
Cette fonction retourne le block associé à une implémentation. (Uniquement disponible depuis iOS 4.3 car nécessite la disponibilité des block Objective-C).

imp_removeBlock
Cette méthode ne supprime pas un block mais supprime l'association qui pourrait être faite entre une implémentation IMP et un block.

imp_implementationWithBlock
Cette méthode permet de créer une implémentation à partir d'un block Objective-C. Cette fonction est très pratique si l'on ne souhaite pas définir de méthodes en dur dans le code (avec prototype, etc...) mais uniquement passer par des block Objective-C qui ont l'avantage d'être plus souples.

objc_setAssociatedObject
Cette fonction permet d'associer un objet à un autre objet.
Cela est très utilisé pour ajouter des propriétés à une catégorie de classe en Objective-C.

objc_getAssociatedObject
Celle-ci retourne l'objet précédemment associé à un autre objet.

objc_removeAssociatedObjects.
Cette fonction supprime l'association ayant précédemment été établie entre deux objets.

Bon code !

Vincent **Saluzzo**
Ingénieur informatique - Expert Technique iOS
http://www.vincentsaluzzo.com

Démarrer avec iOS

2e partie

Dans un précédent article, nous avons étudié les bases d'un projet iOS et comment mettre en place un écran avec une tableView et des cellules personnalisées, tout en récupérant des données depuis un serveur distant. Dans ce nouvel article, nous irons plus loin en ajoutant quelques écrans et en présentant des outils qui facilitent la vie des développeurs iOS.

Pour commencer, si vous n'avez pas lu la première partie de cet article, il est possible d'en récupérer le code source complet depuis cette adresse : https://github.com/xebia-france/programmez-MovieApp/releases/tag/1.0.

CocoaPods

Il existe des milliers de composants et bibliothèques de qualité sur Internet. Ils permettent souvent un gain de temps conséquent lors du développement d'une application. Cependant, les récupérer, gérer les différentes versions, les mises à jour, ou encore les configurer, peut faire perdre un temps précieux ! Afin de pallier ce problème, nous vous conseillons d'utiliser le gestionnaire de dépendances appelé CocoaPods.

Afin de l'utiliser, il suffit de spécifier les dépendances exploitées dans un simple fichier texte pour que CocoaPods résolve les dépendances entre les bibliothèques, récupère les sources, et crée, et/ou, mette à jour, un workspace Xcode afin de compiler votre projet en toute tranquillité.

L'installation de CocoaPods est très facile. Pour cela, il faut s'assurer d'avoir Ruby MRI 2.0.0 ou 1.8.7 et Xcode command line tools installés sur votre machine. Ensuite, il suffit d'exécuter la commande suivante : [sudo] gem install cocoapods. Il est possible d'utiliser la même commande afin de mettre à jour CocoaPods.

Une fois CocoaPods installé, il est maintenant temps de créer le fichier Podfile, qui va lister l'ensemble des dépendances que nous souhaitons utiliser (il est possible de rechercher des dépendances sur http://cocoapods.org). Pour cela, créez un fichier nommé 'Podfile' dans le même répertoire que votre projet puis ajoutez une ligne par dépendance. Par exemple, si votre projet utilise la librairie XBToolKit, il suffit d'ajouter la ligne suivante : pod 'XBToolkit', '~> 0.0.11' (nous verrons par la suite ce qu'est XBToolKit). Enfin, pour récupérer les sources et configurer automatiquement votre projet, il suffit d'exécuter la commande suivante dans le dossier de votre projet : pod install. Si, par la suite, de nouvelles dépendances sont ajoutées au fichier Podfile, il est possible d'exécuter de nouveau "pod install" afin d'installer les dépendances manquantes. Il est également possible de mettre à jour l'ensemble des dépendances en exécutant "pod update". Avec CocoaPods, vous pouvez réaliser des configurations très avancées. L'ensemble des possibilités se trouve sur http://docs.cocoapods.org/index.html.

Une application plus robuste

Dans le chapitre précédent, nous avons créé MADataSource qui nous a permis de récupérer et stocker la liste des films au box-office. Bien que cette approche fonctionne pour une petite application avec un nombre limité d'interactions avec un serveur, elle ne peut être considérée comme appropriée pour des cas plus complexes où une communication plus profonde avec les API serveur s'impose.

Un autre élément sur lequel nous allons travailler est la robustesse de notre code : la précédente version de notre application stockait les données à l'intérieur d'un NSDictionary. NSDictionary est une structure de données suffisamment puissante, mais pas assez fiable si nous travaillons en équipe et que le modèle de données est complexe.

Pour ces raisons notamment, nous allons maintenant introduire un composant open-source, nommé XBToolkit, qui nous permettra de traiter à la fois les requêtes réseau, et les modèles de données, d'une manière simple et robuste. XBToolkit nous donne la possibilité de récupérer un fichier JSON à partir d'un Web Service et de le transformer en objet métier, en utilisant un modèle de données que nous allons fournir à ses fonctions. XBToolkit se base sur des bibliothèques open-source largement adoptées, comme par exemple AFNetworking. Pour installer XBToolkit, nous devons suivre les étapes décrites dans l'introduction de cet article.

Notre fichier Podfile devra contenir les lignes suivantes :

Podfile

```
platform : iOS , '6 .0 '
pod 'XBToolkit', '0 .0.11 '
```

Nous pouvons maintenant clore le fichier, ouvrir le Terminal, et écrire :

Terminal

```
cd <Mon Dossier MovieApp>
pod install
```

Une fois l'exécution terminée, un autre fichier, nommé MovieApp.xcworkspace, apparaît dans notre dossier. Ce fichier sera désormais notre nouveau projet que nous allons ouvrir à chaque fois que nous voudrons travailler sur MovieApp. Comme le suffixe l'indique, MovieApp.xcworkspace est un espace de travail, c'est-à-dire un espace comprenant des références multiples vers d'autres projets. Dans notre cas, MovieApp.xcworkspace comprend notre ancien MovieApp.xcodeproj ainsi qu'un nouveau projet contenant toutes les sources de nos bibliothèques externes, comme XBToolkit.

Création du modèle des données

Comme mentionné précédemment, afin de produire une implémentation robuste, la liste des films que nous téléchargeons à partir du service Web sera transformée en une liste d'instances d'objets métier.

Dans Xcode, nous pouvons créer ces objets dans nos projets en cliquant sur : File -> New -> File -> Objective-C class -> Next

Dans le champ de texte "Class", nous écrirons "MAMovie" et dans "Subclass of", nous saisirons "NSObject".

Afin que XBToolkit puisse convertir le fichier JSON en des instances de classe MAMovie, nous devons préciser que notre objet doit être mappé en utilisant la fonctionnalité de "Mapping" de la bibliothèque. Pour ce faire, nous allons importer XBMappingProvider dans MAMovie.h :

```
#import "XBMappingProvider.h"
```

et nous allons spécifier que MAMovie implémente ce protocole en ajoutant à côté de NSObject <XBMappingProvider>. La ligne de l'interface devra donc être comme suit :

```
@interface MAMovie : NSObject <XBMappingProvider>
```

En dessous de @interface nous pouvons maintenant ajouter toutes les propriétés qui seront remplies automatiquement avec les valeurs JSON :

MAMovie.h

```
@property (nonatomic, strong) NSString *internalBaseClassIdentifier;
@property (nonatomic, strong) NSString *synopsis;
```

```
@property (nonatomic, strong) NSArray *abridgedCast;
@property (nonatomic, strong) NSString *criticsConsensus;
@property (nonatomic, strong) NSString *mpaaRating;
@property (nonatomic, strong) NSString *title;
@property (nonatomic, strong) NSNumber *runtime;
@property (nonatomic, strong) NSNumber *year;
@property (nonatomic, strong) MAPosters *posters;
@property (nonatomic, strong) NSDictionary *ratings;
@property (nonatomic, strong) NSArray *genres;
@property (nonatomic, strong) NSDictionary *links;
```

À l'intérieur de @implementation dans MAMovie.m, nous devons maintenant ajouter le code suivant qui instancie une configuration vide pour le mapping de MAMovie :

MAMovie.m

```
+ (DCParserConfiguration *)mappings {
  DCParserConfiguration *config = [[DCParserConfiguration alloc]
  init];
  return config;
}
```

Une fois que la classe qui contiendra les données de films est créée, nous allons faire de même pour les posters.
Dans Xcode, nous allons ajouter une nouvelle classe Objective-C, du nom de MAPosters, sous-classe de NSObject.
Le code de la classe sera très similaire à celui que nous avons écrit dans MAMovie.

MAPosters.h

```
#import <Foundation/Foundation.h>
#import "XBMappingProvider.h"

@interface MAPosters : NSObject<XBMappingProvider>

@property (nonatomic, strong) NSString *original;
@property (nonatomic, strong) NSString *detailed;
@property (nonatomic, strong) NSString *thumbnail;
@property (nonatomic, strong) NSString *profile;

@end
```

MAPosters.m

```
#import <Foundation/Foundation.h>
#import "MAPosters.h"

@implementation MAPosters

+ (DCParserConfiguration *)mappings {
  DCParserConfiguration *config = [[DCParserConfiguration alloc]
  init];
  return config;
}
@end
```

Revenons à MAMovie.h et ajoutons une autre importation de MAPosters.h (# import "MAPosters.h") pour corriger l'erreur de compilation.
Qu'avons-nous fait ici ? Le code que nous venons d'écrire nous permet de convertir automatiquement toutes les valeurs de "posters" figurant dans le JSON en des instances d'objets MAPosters.

Création d'un client d'API

Nous continuons nos améliorations en supprimant MADataSource.h et MADataSource.m qui seront remplacés en écrivant un code plus propre. Retirez toutes les références à MADataSource de notre projet et, en particulier, de MATableViewController.h et MATableViewController.m.
Notre prochaine étape consistera à créer un client d'API, qui est une classe qui s'occupera de toutes les interactions de notre application avec le serveur. Notre client d'API sera un singleton, c'est à dire une classe à laquelle on peut accéder à partir de différentes parties de notre code sans être instanciée plusieurs fois. Par exemple, dans le développement iOS, UIApplication est un singleton dont l'instance peut être consultée partout avec la syntaxe [UIApplication sharedApplication].
Créons donc notre client d'API, en ajoutant une nouvelle classe nommée MAMovieAppAPIClient, sous-classe de XBHttpClient.
MAMovieAppAPIClient.h sera comme suit.

MAMovieAppAPIClient.h

```
#import "XBHttpClient.h"

typedef void(^MABoxOfficeCallbackBlock)(NSArray *array);

@interface MAMovieAppAPIClient : XBHttpClient

+ (MAMovieAppAPIClient *)sharedClient;
-  (void)downloadBoxOfficeWithCallback:(MABoxOfficeCallback
Block)callback;

@end
```

L'en-tête de la classe expose simplement quelques méthodes publiques que nous allons mettre en œuvre dans MAMovieAppAPIClient.m.
Ouvrons MAMovieAppAPIClient.m et, au-dessus de la ligne @implementation, ajoutons des chaînes de caractères statiques et des importations qui seront utilisées plus tard.

MAMovieAppAPIClient.m

```
#import "XBHttpJsonDataLoader.h"
#import "XBJsonToArrayDataMapper.h"
#import "XBReloadableArrayDataSource.h"
#import "MAMovie.h"

static NSString *kAPI_KEY = @"YOUR_API_KEY";
static NSString *kBoxOfficePath = @"lists/movies/box_office.json";
static NSString *kMAMovieAppAPIBaseURLString = @"http://api.
rottentomatoes.com/api/public/v1.0/";
```

En dessous de @implementation, ajoutons le code suivant :

MAMovieAppAPIClient.m

```
+ (instancetype)sharedClient {
  static MAMovieAppAPIClient *_sharedClient = nil;
  static dispatch_once_t onceToken;
  dispatch_once(&onceToken, ^{
    _sharedClient = [[self alloc] initWithBaseUrl:kMAMovieApp
APIBaseURLString];
  });
  return _sharedClient;
}

- (id)initWithBaseUrl:(NSString *)url {
```

```
  self = [super initWithBaseUrl:url];
  if (!self) {
    return nil;
  }

  return self;
}
```

Vu que notre client d'API est un singleton, la fonction sharedClient sera le point d'entrée unique de notre exemple. Les lignes que nous venons d'écrire, et en particulier la fonction dispatch_once, précisent que le code à l'intérieur ne doit être effectué qu'une seule fois, de sorte qu'une seule instance du client API puisse être créée.
Ajoutons la méthode "rottenTomatoesUrlForPath" permettant de générer une URL compatible avec les Rotten Tomatoes services Web.

MAMovieAppAPIClient.m

```
- (NSString *)rottenTomatoesUrlForPath:(NSString *)relativePath {

  // Retrieve the current user country code
  NSLocale *locale = [NSLocale currentLocale];
  NSString *countryCode = [locale objectForKey: NSLocaleCountry
Code];

  // Create the params dictionary
  NSDictionary *params = @{@"country": countryCode,
                @"limit": @20};

  // Create a new mutable string starting with our base URI
  NSMutableString *urlString = [NSMutableString stringWith
String:@""];

  // Append the relative path and our api key
  [urlString appendString:relativePath];
  [urlString appendFormat:@"?apikey=%@",kAPI_KEY];

  // Append all params in the dictionary as key = obj
  [params enumerateKeysAndObjectsUsingBlock:^(id key, id obj,
 BOOL *stop) {
    [urlString appendFormat:@"&%@=%@", key, obj];
  }];

  NSLog(@"Loading data from : '%@'", urlString);

  return urlString;
}
```

Et enfin, ajoutons la méthode downloadBoxOfficeWithCallback, comme suit :

MAMovieAppAPIClient.m

```
-  (void)downloadBoxOfficeWithCallback:(MABoxOfficeCallback
Block)callback {

  // Instantiate an dataLoader from your HTTP client and a given
resourcePath:
  XBHttpJsonDataLoader *boxOfficeDataLoader = [XBHttpJsonData
Loader dataLoaderWithHttpClient:self resourcePath:[self rotten
TomatoesUrlForPath:kBoxOfficePath]];

  // Instantiate an dataMapper, allowing the response to be
deserialized to a given class (e.g. MAMovie):
  XBJsonToArrayDataMapper *boxOfficeDataMapper = [XBJsonToArray
DataMapper mapperWithRootKeyPath:@"movies" typeClass:[MAMovie
 class]];

  // Create the data source from the dataLoader and the dataMapper:
  XBReloadableArrayDataSource *dataSource = [XBReloadableArray
DataSource dataSourceWithDataLoader:boxOfficeDataLoader data
Mapper:boxOfficeDataMapper];

  [dataSource loadDataWithCallback:^{
    if (callback) {
      callback(dataSource.array);
    }
  }];
}
```

downloadBoxOfficeWithCallback est une méthode simple mais puissante qui, avec un nombre de lignes succinct, prend en charge les étapes suivantes :

1. Création d'une instance s'occupant de gérer le téléchargement du box-office depuis le Web Service en utilisant XBHttpJsonDataLoader.
2. Mapping des films au box-office vers notre objet MAMovie.
3. Création d'un dataSource qui relie téléchargement et mapping.
4. Lancer le téléchargement et déclencher la méthode de callback.

Grâce à cette implémentation, il est vraiment facile d'étendre les fonctionnalités du client d'API. Si nous souhaitons récupérer des données supplémentaires depuis les services Web de Rotten Tomatoes, il suffit de créer une méthode similaire à downloadBoxOfficeWithCallback à l'intérieur de notre classe de client d'API.

Affichage des données

Maintenant que nous avons créé le client API, nous devons rétablir les fonctionnalités de notre application.
Revenons à MATableViewController.m et :

1. importons "MAMovieAppAPIClient.h", "MAMovie.h" et "UIImageView+AFNetworking.h",
2. ajoutons les propriétés privées "movies" entre @interface et @end, comme suit :

```
@property (nonatomic, strong) NSArray *movies;
```

Dans viewDidLoad, nous allons ajouter les lignes suivantes.

MATableViewController.m

```
[[MAMovieAppAPIClient sharedClient] downloadBoxOfficeWithCall
back:^(NSArray *array) {
    self.movies = array;
    [self.tableView reloadData];
  }];
```

Ce code exécute la méthode de downloadBoxOffice de notre client d'API singleton, et, une fois l'opération terminée, s'occupe de stocker les résultats à l'intérieur de la propriété movies et de recharger la tableView.
Dans la méthode :

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRows
InSection:(NSInteger)section
```

Nous allons remplacer l'implémentation existante par return [self.movies count]; Enfin, nous remplaçons l'implémentation de cellForRowAtIndexPath par la suivante :

MATableViewController.m

```
static NSString *CellIdentifier = @"MovieCell";

MAMovieCell *cell = [tableView dequeueReusableCellWithIdentifier:
CellIdentifier forIndexPath:indexPath];

// Retrieve the MAMovie object
MAMovie *movie = self.movies[indexPath.row];
cell.titleLabel.text = movie.title;
cell.subtitleLabel.text = [movie.year stringValue];

// The syntax movie.posters.thumbnail allows to retrieve the
thumbnail string from the object MAPosters contained in MAMovie
[cell.thumbnail setImageWithURL:[NSURL URLWithString:movie.
posters.thumbnail] placeholderImage:nil];

return cell;
```

Avec ces dernières modifications, nous nous sommes finalement débarrassés de NSDictionary, et, grâce aux objets métier, nous pouvons maintenant bénéficier d'un meilleur contrôle sur les types de données des propriétés, avoir un retour immédiat sur les erreurs grâce aux contrôles de compilation et - last but not least - enfin bénéficier de l'autocomplétion de code !

Nouveau View Controller et transition

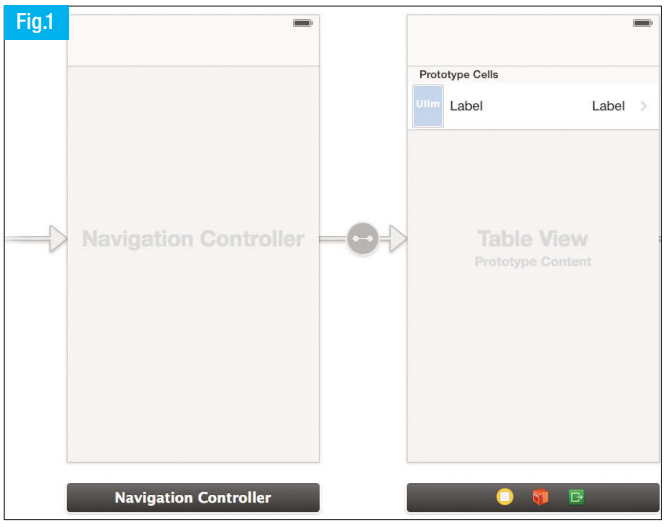
À ce stade de l'application, nous souhaitons créer un nouveau View Controller dont le rôle sera d'afficher les détails d'un film que l'on sélectionne sur la page d'accueil. Pour cela, commençons par créer les fichiers relatifs au nouveau controller : File > New > File puis sélectionnez "Classe Objective-C". Nous allons appeler cette nouvelle classe "MAMovieDetailsViewController".

Ce controller devant afficher les détails d'un film, il nous faut déclarer une nouvelle propriété qui représente le film en question :

MAMovieDetailsViewController.m

```
@interface MAMovieDetailsViewController : UIViewController

@property (nonatomic, strong) MAMovie *movie;
```




```
@end
```

Il est maintenant temps de passer dans le storyboard de l'application afin de réaliser la transition entre les cellules de la home page, et notre page détails. Une fois dans le storyboard, commençons par ajouter un "Navigation View Controller" avec le controller de la page d'accueil pour root view controller. Cela nous permettra de pouvoir naviguer entre différents controller : Fig.1.

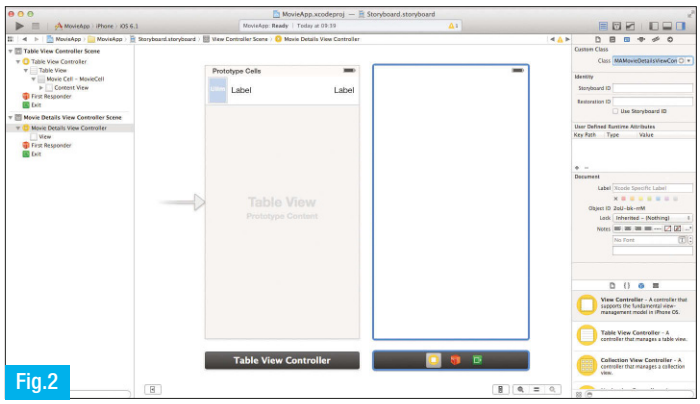
Ensuite, faisons un drag & drop d'un View Controller à côté du controller de la page d'accueil, puis dans l'entity inspector, spécifiez qu'il s'agit d'un "MAMovieDetailsViewController" : Fig.2.

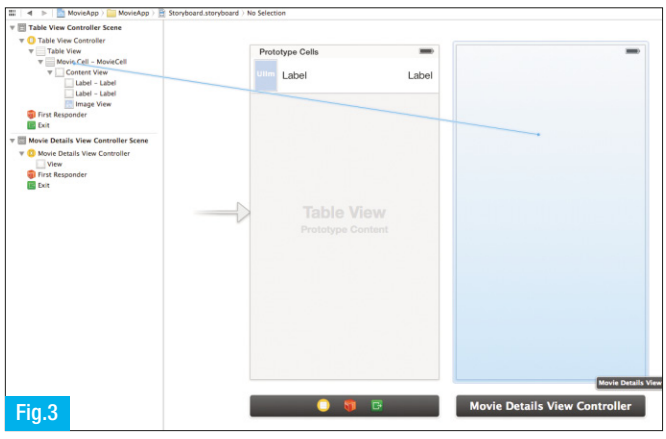
Afin de réaliser la transition entre les cellules et ce controller, tirez un trait entre la cellule et le controller tout en maintenant la touche CTRL enfoncée : Fig.3.

Une popup va s'ouvrir avec différentes options qui permettent de choisir quel type de transition effectuer. Choisissons "push". Nous pouvons à présent voir dans le storyboard une flèche entre nos deux viewController qui représente le fait qu'une transition est possible entre ces deux derniers Fig.4.

Afin d'afficher les détails d'un film, le View Controller MAMovieDetailsViewController a besoin d'une instance de MAMovie. Pour cela, il faut lui passer l'instance en paramètre lors de la transition. Nous avons donc besoin d'identifier dans le code la transition, plus communément appelée un "segue". Toujours dans le storyboard, sélectionnez la flèche entre les deux viewController, puis renseignons un identifiant de segue dans l' "Attributes inspector" : appelons le "MAShowMovieDetails" : Fig.5.

Passons maintenant dans "MATableViewController" afin de passer l'objet métier MAMovie au View Controller MAMovieDetailsViewController. Il est possible d'interagir avec le controller cible lorsqu'une transition est déclenchée. Afin de le faire, il faut overrider la méthode prepareForSegue:sender:. Voici à quoi ressemble notre implémentation :





MAMovieDetailsViewController.m

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
  if ([[segue identifier] isEqualToString:@"MAShowMovieDetails"]) {
     NSIndexPath *indexPath = [self.tableView indexPathForSelectedRow];
     MAMovie *aMovie = [self.movies objectAtIndex:indexPath.row];

     MAMovieDetailsViewController *vc = [segue destinationViewController];
     vc.movie = aMovie;
  }
}
```

Une fois cela implémenté, votre MAMovieDetailsViewController sera instancié et affiché tout en ayant une référence vers un MAMovie.

Affichage des détails du film

Notre objet MAMovie possède plusieurs propriétés dont les valeurs ont été renseignées lorsque nous avons récupéré la liste du box-office, telles que le titre, la durée, le synopsis... Ce sont ces informations que nous allons mettre en avant sur la fiche de détails. Nous avons donc besoin de plusieurs éléments visuels que nous allons ajouter dans MAMovieDetailsViewController.h

MAMovieDetailsViewController.h

```
@property (nonatomic, weak) IBOutlet UIView *headerView;
@property (nonatomic, weak) IBOutlet UIImageView *posterImageView;
@property (nonatomic, weak) IBOutlet UILabel *genreLabel;
@property (nonatomic, weak) IBOutlet UILabel *runtimeLabel;
@property (nonatomic, weak) IBOutlet UIImageView *ratingImageView;
@property (nonatomic, weak) IBOutlet UILabel *ratingLabel;
@property (nonatomic, weak) IBOutlet UITextView *synopsisTextView;
```

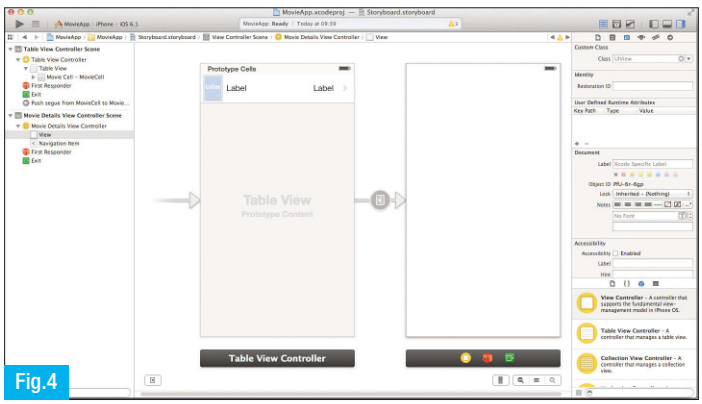
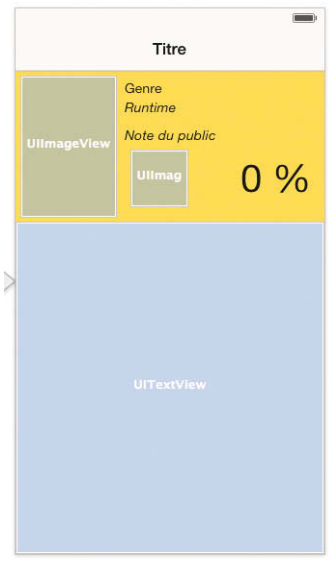


Fig.4



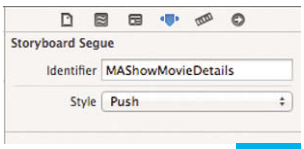


Fig.5

Fig.6

Afin de pouvoir lier ces éléments directement via le storyboard, nous leur ajoutons la propriété IBOutlet. Nous n'avons plus qu'à les disposer sur le storyboard, et voici le très beau design que nous avons réalisé : Fig.6. Afin de lier ces éléments aux propriétés de notre controller, sélectionnons-le dans la barre de gauche (1) puis choisissons l'outlet collection dans l'inspecteur (2), avant de lier chaque propriété à un élément du board (3) Fig.7. Maintenant que tout est prêt, il ne reste qu'à remplir notre vue avec les données du film. Pour cela, nous allons créer une méthode updateView dans MAMovieDetailsViewController.m

MAMovieDetailsViewController.m

```
- (void)updateView {

  // If the view is not yet loaded, no outlets will be set so
  get out of here !
  if (!self.isViewLoaded) {
    return;
  }

  // Update title
  self.title = [NSString stringWithFormat:@"%@ (%@)", self.movie.
title, self.movie.year];

  // Update Poster image
  [self.posterImageView setImageWithURL:[NSURL URLWithString:
self.movie.posters.profile]];

  // Update genre
  self.genreLabel.text = [self.movie.genres componentsJoined
ByString:@", "];

  // Update runtime
  self.runtimeLabel.text = [NSString stringWithFormat:@"%@ min",
  self.movie.runtime];

  // Update synopsis
```

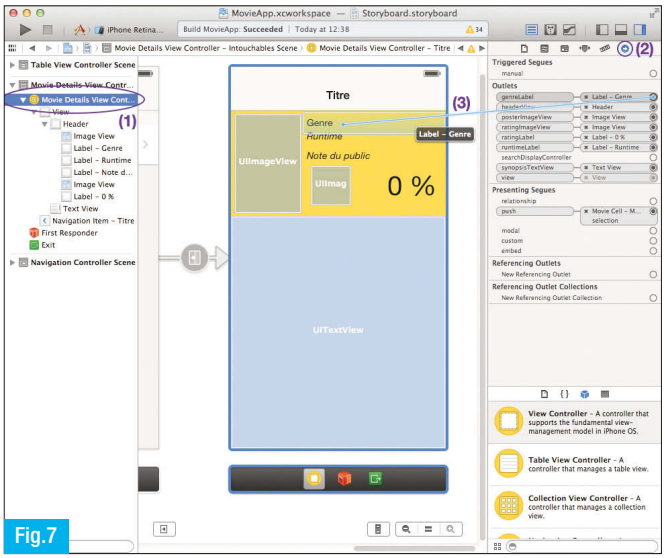


Fig.7

```
  self.synopsisTextView.text = self.movie.synopsis;

  // Update rating and start animations
  int ratings = [self.movie.ratings[@"audience_score"] intValue];
  self.ratingLabel.text = @"0 %";
  self.ratingImageView.image = (ratings < 50) ? [UIImage image
Named:@"rotten"] : [UIImage imageNamed:@"fresh"];

  self.ratingImageView.alpha = .0;
  [UIView animateWithDuration:kRatingAnimationDuration animations:^{
    self.ratingImageView.alpha = 1;
  }];

  [NSTimer scheduledTimerWithTimeInterval:kRatingAnimationDuration
/ratings target:self selector:@selector(updateRatingText:) user
Info:nil repeats:YES];
 }

-(void) updateRatingText:(NSTimer*)timer {
  int rating = [self.ratingLabel.text intValue] + 1;

  self.ratingLabel.text = [NSString stringWithFormat:@"%d %%",
rating];

  // Stop the timer if our rating has the final value
  if (rating >= [self.movie.ratings[@"audience_score"] intValue]) {
    [timer invalidate];
  }
}
```

Afin de rendre les choses un peu plus fun, nous avons ajouté deux petites animations pour afficher la note du public. Tout d'abord, nous affichons soit une tomate, soit un amas vert en fonction de la note (nous restons fidèles au design RottenTomatoes sur ce point !) en modifiant sa propriété alpha afin qu'elle apparaisse en fondu.

MAMovieDetailsViewController.m

```
self.ratingImageView.alpha = .0;
[UIView animateWithDuration:kRatingAnimationDuration animations:^{
  self.ratingImageView.alpha = 1;
}];
```

Puis nous voulons faire défiler les chiffres de 0 jusqu'à la valeur de la note réelle. Pour cela, nous créons un timer qui va appeler la méthode updateRatingText toutes les 1/note seconde.

MAMovieDetailsViewController.m

```
[NSTimer scheduledTimerWithTimeInterval:kRatingAnimationDuration
/ratings target:self selector:@selector(updateRatingText:) user
Info:nil repeats:YES];
```

Il ne nous reste plus qu'à appeler la méthode updateView une fois que l'objet movie a été assigné à notre controller. Mais, si vous avez été attentifs, vous avez remarqué que nous utilisons une propriété "genres" sur notre objet MAMovie, alors que cette clé n'a pas été envoyée lors de la récupération du box-office. En fait, pour une raison un peu obscure, cette propriété n'est accessible que lorsque nous requêtons RottenTomatoes pour un film spécifique. Alors comment l'obtenir ? Et bien nous allons effectuer un appel supplémentaire pour récupérer les informations manquantes. Nous retournons donc dans MAMovieAppAPIClient et créons une méthode qui prend en paramètre un objet MAMovie, effectue la requête, met à jour les infos, puis appelle un callback s'il a été défini.

MAMovieAppAPIClient.h

```
- (void) updateMovieData:(MAMovie *)movie withCallback:(void
  (^)(void))callback;
```

MAMovieAppAPIClient.m

```
- (void) updateMovieData:(MAMovie *)movie withCallback:(void
(^)(void))callback {

  // Get the path for the current movie
  NSString *link = [movie.links[@"self"] stringByReplacing
OccurrencesOfString:kMAMovieAppAPIBaseURLString withString:@""];

  // Instantiate an dataLoader from your HTTP client and a given
resourcePath:
  XBHttpJsonDataLoader *dataLoader = [XBHttpJsonDataLoader data
LoaderWithHttpClient:self resourcePath:[self rottenTomatoes
UrlForPath:link]];

  // Instantiate an dataMapper, allowing the response to be
deserialized to a given class (e.g. MAMovie):
  XBJsonToObjectDataMapper * dataMapper = [XBJsonToObjectData
Mapper mapperWithRootKeyPath:nil typeClass:[MAMovie class]];

  // Create the data source from the dataLoader and the dataMapper:
  XBReloadableObjectDataSource *dataSource = [XBReloadable
ObjectDataSource dataSourceWithDataLoader:dataLoader data
Mapper:dataMapper];

  [dataSource loadDataWithCallback:^{

    MAMovie *updatedMovie = (MAMovie*)dataSource.object;
    movie.genres = [NSArray arrayWithArray:updatedMovie.genres];

    if (callback) {
      callback();
    }
  }];
}
```

Et maintenant, pour conclure l'affichage correct de nos détails, nous appelons updateView une fois les nouvelles informations renseignées. Dans MAMovieDetailsViewController, définissons le setter de la propriété movie :

MAMovieDetailsViewController.m

```
- (void)setMovie:(MAMovie *)movie {
  _movie = movie;

  [self.movie updateInfoWithCallback:^{
    // Call updateView on main thread because it uses graphical
elements, which can ONLY be used on the main thread
    [self performSelectorOnMainThread:@selector(updateView) with
Object:nil waitUntilDone:NO];
  }];
}
```

Simone **Civetta**, Nicolas **Thenoz**, Martin **Moizard**,
consultants chez Xebia

Grails : un framework plein de potentiel

Grails est un framework web Open Source dit de haute productivité et agile basé sur Groovy et Java. Pourquoi un autre framework de développement Java? Qu'apporte Grails de plus que les autres? Dans cet article, nous répondrons à ces questions en réalisant une première application simple : it-resto

En 2005, Graeme Rocher a initié ce projet pour apporter un équivalent de Ruby on Rails au monde Java. Le nom Grails est d'ailleurs une contraction de Groovy, langage sur lequel il est basé, et Rails... L'ensemble devant former Grails (Les Graals anglais, d'où le choix du logo).

Aujourd'hui, nous en sommes à la version 2.3.7 (sortie en mars 2014).

Mais, du coup, Grails, c'est quoi ?

Il s'agit d'un framework MVC Open Source qui fonctionne sur une JVM (Java Virtual Machine). Sa philosophie tourne autour des points suivants :

- Java-like dynamic language : les scripts Groovy sont compilés en byte-code, permettant de profiter de la puissance de Java. Il s'appuie sur son écosystème en utilisant Spring, Hibernaten etc.
- Convention over Configuration
- Don't Repeat Yourself (DRY)
- Model Driven Architecture (MDA) : une partie du code peut être générée depuis le modèle.
- Prototypage : grâce au scaffolding, il est très rapide de générer un premier prototype (même incomplet) de l'application.

Avant de démarrer

En pré-requis, une version du JDK (Java Developpement Kit), version 1.6 minimum doit être correctement configurée.

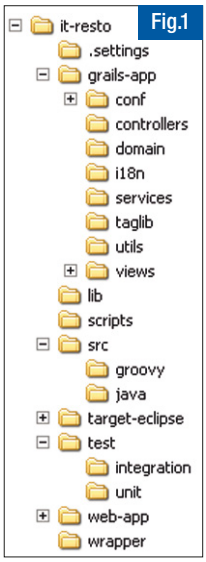
Pour installer Grails, il faut télécharger la dernière version (ou version désirée sur le site http://grails.org) et extraire le zip à l'emplacement de son choix. La variable GRAILS_HOME indique le lieu de l'extraction. La variable PATH est mise à jour, pour y ajouter le répertoire bin de l'extraction. En lançant la commande *grails –version*, on sait très vite si l'installation a réussi.

Pour les IDE, il y a du choix. A savoir que SpringSource propose le Groovy/Grails Tools Suite (GGTS) basé sur Eclipse.

Définition du projet

Pour les besoins pédagogiques et démonstratifs de l'article, une application, it-resto, va être développée tout au long de l'article.

It-resto permet aux utilisateurs de participer à un événement. Pour chaque événement, il est possible de voter pour un ou plusieurs restaurants, parmi une liste précise. L'ensemble des votes permet de définir le restaurant qui accueillera l'événement. Un cas d'utilisation simple serait, au sein d'un service dans une entreprise, de permettre à un groupe de collègues de sélectionner le lieu où ils vont déjeuner.



Arborescence d'un projet Grails

L'utilisateur pourra :

- S'authentifier,
- Créer un évènement,
- Participer à un évènement (sans limitation de droit), en votant (une seule fois) pour un ou plusieurs restaurants dans un événement.

L'administrateur pourra gérer (Create, Update, Read et Delete) les utilisateurs, les restaurants, les événements et les votes.

Les événements passés et leurs votes associés seront automatiquement supprimés toutes les nuits.

Premier pas avec Grails

Voilà, tout est prêt. Il est temps de se lancer. Pour créer le projet it-retso, il faut lancer la commande *grails create-app it-resto*

Un squelette d'arborescence (Figure 1) du projet est créé dans le répertoire it-resto.

Le répertoire grails-app est le plus important. C'est ici que vont se trouver les principales ressources du projet :

Conf: ensemble des fichiers de configuration pour Spring, Hibernate et le projet en cours.

Controllers: le nom ne laisse que peu de surprise. Le répertoire contient les classes contôleurs.

Views: il s'agit des vues du projet, au format gsp.

Le répertoire lib stockera les différentes librairies du projet, qu'elles soient ajoutées via maven, ivy ou manuellement.

src contient les sources java et/ou groovy

Pour finir, test... comme son nom l'indique, on y retrouve les tests unitaires, mais aussi les tests d'intégration.

Pour tester que tout s'est bien déroulé, il est possible de lancer le projet (*grails run-app*) et d'accéder au projet (http://localhost:8080/it-resto/)

Par défaut, Grails s'exécute dans Tomcat.

Mise en place du modèle

L'un des principes de base de Grails est d'être orienté modèle. La première étape est donc de générer le modèle de données.

Notre modèle possède quatres éléments distincts:

- Restaurant : modélisation des restaurants à sélectionner par les utilisateurs dans un événement.
- User : utilisateur enregistré.
- Event : événement de base, qui englobera les votes des utilisateurs
- Vote : sur chaque événement, un utilisateur pourra sélectionner un ou plusieurs restaurants s'il souhaite y participer.

Pour aller plus vite, passons en mode interactif avec Grails. Ce mode permet de travailler directement dans la console Grails, sans avoir à relancer le framework à chaque commande. De plus, les changements sont pris en compte dynamiquement. Lancez Grails: *grails*
Il faut générer les quatre éléments du modèle de données (le domain).

```
create-domain-class it.resto.restaurant
create-domain-class it.resto.user
create-domain-class it.resto.event
create-domain-class it.resto.vote
```

A chaque fois, deux fichiers sont créés :

- Le premier, dans grails-app/domain/it/resto/*.groovy, pour la définition du domain
- Le second, dans test/unit/it/resto/*Spec.groovy pour les tests unitaires

En fonction du besoin du projet, les différents domains doivent être complétés.
Pour le domain Restaurant, on aura

```
class Restaurant {
    String name

  static constraints = {
    name blank: false, unique:true
  }
}
```

L'objet *it.resto.Restaurant* n'est défini que par son nom (*name*). Ensuite, dans le bloc *constraints*, on peut définir toutes les contraintes sur les différents champs. Dans notre cas, le nom du restaurant sera non null (par défaut dans les domains Grails), non vide (*blank:false*) et unique (*unique:true*). Bien que plus complète, la déclaration d'un User est assez similaire.

```
class User {
    String name
    String lastName
    String login
    String password
    String email

  static constraints = {
    name blank: false
    lastName blank: false
    login size: 5..15, blank: false, unique: true
    password size: 5..15, blank: false
    email email: true
  }
}
```

Cette fois, les contraintes sont plus importantes. On notera surtout que la taille du login et du password doivent être comprises entre 5 et 15 caractères (*size: 5..15*). Le champs email quant à lui doit répondre aux contraintes d'un email (*email: true*). Maintenant, passons à la classe *it.resto.Event*, qui devra être liée avec plusieurs votes.

```
class Event {
    String name
    Date eventDate
    User owner
    static hasMany = [votes: Vote]

  static constraints = {
    name blank: false
  }
}
```

En regardant l'ensemble des contraintes, on se rend compte que tous les champs sont obligatoires. La relation de type 1 N entre l'Event et les Vote est défini avec le mot clef *hasMany*, qui déclare une liste de type *it.resto.Vote*, accessible par le champ *votes*. Pour chaque Event, on garde également une trace de l'utilisateur qui crée l'Event, dans le champ *owner*, qui permet de faire une relation de type 1 1 entre un Event et un User.
Terminons avec le Vote qui va relier un Event, un User et la liste des Restaurants que le participant aura sélectionnés.

```
class Vote {
    User user
    static hasMany = [restaurants: Restaurant]
    static belongsTo = [event:Event]

  static constraints = {
  }
}
```

La classe *it.resto.User* définit dans le Vote représente la relation lien direct de type 1 1, ce qui signifie qu'un vote est attaché à un seul et unique User. Les références vers les restaurants sélectionnés par l'utilisateur sont déclarées avec le mot clef *hasMany*. Un Vote est relié à un ou plusieurs Restaurant. L'Event est déclaré avec *belongsTo*, indiquant que le Vote lui appartient. On retrouve son équivalent en *hasMany* dans la classe *it.resto.Event*. En supprimant un Event, les Vote associés seront également supprimés. Dans tous les cas, l'identifiant technique *id* est implicite.
Voilà, nous sommes prêts... GORM (Grails Object Relationnal Mapping) va faire le reste. Il s'agit d'une surcouche au framework hibernate qui génère automatiquement toute une série de méthodes telles que load, save, exist... Mais, également une partie plus «magique»; des méthodes de recherches comme findBy* qui sont définies en fonction de la classe.
Par exemple, pour la classe *it.resto.Restaurant*, on retrouvera la findByName ou findByNameLike (voir Tableau 1 - Exemple de méthode applicable avec le findBy).

	findBy...
InList	Dans une liste de valeurs données
LessThan	Inférieur à une valeur donnée
LessThanEquals	Inférieur ou égal à une valeur donnée
GreaterThan	Supérieur à une valeur donnée
GreaterThanEquals	Supérieur ou égale à une valeur donnée
Like	Recherche d'un pattern dans une chaîne
Ilike	Similaire au Like, mais insensible à la casse
NotEqual	Différent de la valeur donnée
InRange	Se situant entre deux valeurs ordonnées
Rlike	Similaire au Like, mais avec la possibilité de réaliser des expressions régulières
Between	Se situant entre deux valeurs
IsNotNull	Contenu dans null
IsNull	Contenu null

Tableau 1 - Exemple de méthode applicable avec le findBy

Mise en place des premiers tests

En parallèle de la création des différentes classes de domain, Grails a également généré des classes de test. Elles se présentent toutes de la même manière avec une méthode *setup*, qui sera jouée avant chaque méthodes de test, ainsi qu'une méthode *cleanup* exécutée après le test.
Il ne reste plus qu'à écrire les méthodes de test pour valider le bon fonctionnement des classes du domain.
Ces méthodes doivent être écrites en plusieurs sections, connues sous le nom de given-when-then.
La section *given* décrit l'état du système avant de débuter le scénario. Ce sont les conditions préalables.
La section *when* est le déroulement du scénario à spécifier.
La section *then* va décrire et tester les changements attendus suite au scénario réalisé dans la section *when*.
Un test sur la création et la validation du domain Restaurant ressemblera à ceci.

```
void 'test constraint Restaurant'() {
  given:
    mockForConstraintsTests Restaurant

  when: 'the restaurant name is null'
    def restaurant = new Restaurant()
    restaurant.name = null

  then: 'validation should fail'
    !restaurant.validate()
    restaurant.hasErrors()
}
```

Dans la section *given*, la seule chose définie est le *mockForConstraintsTests* qui mocke la classe *it.resto.Restaurant* (toute la machine Grails n'est pas lancée durant les tests). Ce mock ajoute la méthode *valide()* à la classe, mais permet aussi de détecter plus simplement des erreurs avec la propriété *errors.* De plus, les messages sont "allégés" et plus simples à analyser.
Par contre, il y a une limitation. Il ne faut pas oublier que le contexte d'exécution est un test unitaire, et non pas d'intégration. Il n'y a pas d'ajout de méthodes au runtime par GORM par exemple (findBy*, etc).
Dans la section *when*, c'est la zone d'exécution du scénario de test. Ici, c'est un objet *it.resto.Restaurant* qui est créé, mais le nom (*name*) est null (explicitement déclaré).
Dans la section *then*, on valide l'état de sortie du scénario. La méthode *valide()* va déterminer si toutes les contraintes sont respectées.
Toutes les instructions de la section *then* doivent être vraies, le contraire indiquant une erreur du test. Pour simplifier un peu plus la détection des erreurs, la méthode *hasErrors()* indique si des erreurs ont été rencontrées, puis le message est disponible en fonction du champ en erreur. Dans cet exemple, ce sera *restaurant.errors['name']*.
Pour démarrer le test, lancer la commande *test-app* (ou *grails test-app*, si vous avez quitté le mode interactif). A la fin, un rapport est disponible dans \target\test-reports\.
Il est recommandé de réaliser cette étape au fur et à mesure de la création des classes du domain.

Peuplement de la base

Le squelette de l'application a été réalisé et le modèle de données est prêt (définition des objets domains ainsi que les tests unitaires sur les contraintes). Avant de réaliser les premiers écrans et pour avoir des éléments à y afficher, il faut peupler la base de données. Cela se fait simplement, depuis le fichier grails-app/conf/BootStrap.groovy qui contient la méthode *init* exécutée au lancement de l'application et la méthode *destroy* quand l'application est arrêtée.
Il suffit de mettre en place le code suivant dans la méthode *init*:

```
if (!Restaurant.count()) {
    print 'Create'
      new Restaurant(name: 'King Croissant').save(failOnError: true)
      new Restaurant(name: 'Les trois mi-temps').save(failOnError: true)
}
```

Restaurant.count() dénombre les restaurants en base. Cette condition évite de créer systématiquement les mêmes éléments à chaque lancement de l'application (ce qui arrive régulièrement en phase de développement).
new Restaurant(name: 'King Croissant').save(failOnError: true) crée un Restaurant et le stocke en base.
La même chose doit être faite pour les autres éléments du modèle : User, Event et Vote.
Pour la base de données sous jacente, Grails propose une connexion à la base H2 en natif, en mode stockage en mémoire et une console d'administration accessible via *http://localhost:8080/it-reso/console*. L'identifiant par défaut est admin et il n'y a pas de mot de passe.
Attention ! Si vous utilisez cette base pour votre application de production, n'oubliez pas de changer le mot de passe admin et de bloquer l'accès à la console, sinon cela revient à exposer tout le contenu de la base sur Internet.
La configuration de la base se fait dans le fichier *grails-app/conf/DataSource.groovy*, au niveau de la propriété *environments.development.datasource*

```
    dataSource {
      dbCreate = «create-drop»
      url = «jdbc:h2:mem:devDb;MVCC=TRUE;LOCK_TIMEOUT=10000»
    }
```

Initialement, la propriété dbCreate est à create-drop, mais elle peut prendre d'autres valeurs, selon le besoin:
create: supprime les tables, les index, etc. avant de recréer le schéma au démarrage
create-drop: semblable à create, mais en supprimant les tables à l'arrêt de l'application.
update: met à jour la table en créant les tables et les index manquants, sans perdre les données existantes.
validate: ne fait aucun changement dans la base de données, mais compare le schéma existant avec le schéma configuré dans l'application (objet domain) et génère un rapport.
Attention, create-drop et create sont à manier avec précaution en dehors d'un environnement de développement. On retrouve des configurations équivalentes pour les environnements de test et de production.

Premier écran

Le premier écran à générer est l'écran principal de l'application listant les événements en cours. Les écrans sont gérés à partir des contrôleurs, que Grails peut générer très rapidement avec la commande *create-controler*.

```
create-controller it.resto.Event
```

Le fichier EventController.groovy est créé dans grails-app/controllers/it/resto (et toujours son pendant EventControllerSpec.

groovy pour les tests dans test/unit/it/resto). En lançant (*run-app*) et testant (http://localhost:8080/it-resto) l'application, on constate qu'un nouveau contrôleur est accessible dans la liste proposée. Pour l'instant, il ne fait pas grand chose. Dans un contrôleur, chaque méthode correspond à une URI de l'application.
La première page doit renvoyer la liste des événements (normalement, deux chargés au démarrage par l'initialisation du bootstrap). Cette méthode est *listEvent* et définie de la façon suivante:

```
def listEvent() {
  render Event.listOrderByName()
}
```

La méthode renvoie tous les événements disponibles ordonnés par nom (*Name*). Un premier test rapide (en plus du test unitaire) est de vérifier que la liste est bien renvoyée dans un navigateur :
http://localhost:8080/it-resto/event/listEvent
Bien... mais, pas très présentable. C'est là qu'intervient le GSP (Groovy Server Page), qui est la partie view de Grails, assez proche du JSP ou de l'ASP. Comme décrit en introduction, le framework Grails préfère la convention à la configuration, donc en créant dans grails-app/views/event/ la gsp listEvent.gsp, le lien entre vue et contrôleur automatique. Enfin, pour qu'il soit totalement pris en compte, modifiez la méthode listEvent de la façon suivante :

```
def listEvent() {
  [events: Event.listOrderByName()]
}
```

Le contrôleur ne fait plus le rendu directement, mais passe par le paramètre *events* à la vue.
Au niveau de la vue, la liste se génère de la façon suivante :
<h2>Liste des événements en cours</h2>

```
<ul style=»font-size: 100%»>
 <g:each var=»event» in=»${events}»>
   <li><a  href=»/it-resto/event/voteEvent?eventId=${event.
id}»>${event.name}</a>, proposé par ${event.owner.name}</li>
 </g:each>
</ul>
```

En vérifiant, nous constatons que la page est correctement rendue dans le navigateur.

Et les taglib...

Maintenant que nous avons la liste des événements, il faudrait aussi pouvoir afficher la date. Mais, plutôt que de rechercher des méthodes complexes de formatage de la date, nous allons réaliser une tag-lib de formatage de la date à utiliser directement dans la vue.
Le fonctionnement s'avère assez proche d'un contrôleur. La taglib est créée par une simple commande: *create-tag-lib it.resto.tag.Date*
Comme précédemment, deux fichiers sont créés:

- Le premier pour la taglib: grails-app/taglib/it/resto/tag/DateTagLib.groovy
- Le second pour les tests unitaires: test/unit/it/resto/tag/DateTagLibSpec.groovy

Tout comme les contrôleurs, c'est le nom de la méthode qui détermine le nom du tag.
Si nous souhaitons un tag de ce style, ou *format* indique le gabarit de la date de sortie souhaitée :

```
<g:dateFormat format=»hh:mm» date=»${event.eventDate}»/>
```

La tag-lib devra être de cette forme :

```
def dateFormat = { attrs, body ->
  if (attrs.date != null) {
    out << new java.text.SimpleDateFormat(attrs.format).format
(attrs.date)
  }
  else {
    out << «»
  }
}
```

Si l'attribut date est vide, une chaîne vide est renvoyée, sinon c'est une date au format souhaité.
La prise en compte des modifications étant faite à chaud, la vue peut être modifiée et le rendu vérifié instantanément.
Modification à apporter dans listEvent.gsp pour prendre en compte le nouveau tag créé :

```
<li><a href=»/it-resto/event/voteEvent?eventId=${event.id}»
>${event.name}</a>, proposé par ${event.owner.name} (le <g:
dateFormat format=»dd/MM/yyyy» date=»${event.eventDate}»/>
à <g:dateFormat format=»hh:mm» date=»${event.eventDate}»/>)</li>
```

Mise en place d'un système d'authentification

Lors de la première page mise en place, il faut penser à la protéger (ainsi que les prochaines pages) en identifiant les utilisateurs à leurs arrivées sur le site. Comment ? En utilisant un filtre sur les requêtes qui teste sur chacune s'il y a bien eu authentification.
La création d'un filtre se fait avec la commande *create-filters*. Dans notre cas, pour le filtre d'authentification, on aura:
grails create-filters it.resto.Authenticate
Le filtre est grails-app/conf/it/resto sous le nom AuthenticateFilters.groovy et les tests unitaires sont disponibles dans test/unit/it/resto/AuthenticateFiltersSpec.groovy
Tous les filtres doivent être définis dans un bloc filters (*def filters* = {...}) avec un nom et une portée. Le nom du filtre est le nom de la méthode et le scope correspondant au paramètre de la méthode.
Par exemple, dans notre cas d'authentification, nous souhaitons que tous les accès aux différents contrôleurs soient vérifiés. Le bloc filters est défini de cette façon :

```
def filters = {
  loginCheck(controller: '*', action: '*') {
    before = {
      if (!session.user && !actionName.equals('login')) {
        redirect(controller: «Event», action: «login»)
        return false
      }
    }
  }
}
```

Le nom du filtre est loginCheck et il s'applique à tous les contrôleurs et toutes leurs actions.
Dans cet exemple, l'interceptor est avant l'action. Le fait de retourner false bloque tous les autres filtres. La condition teste qu'il n'y a pas de champ user en session (pas encore identifié) et si l'action n'est pas login (Il

faut éviter de bloquer la page de login également). Si personne n'est authentifié, le filtre redirige vers l'action *login* du controlleur Event (seul contrôleur actuellement existant).
Cette action est associée à une page de login, qui verifiera la corrélation entre le login et le password.

```
if (params.login != null) {
  def user = User.findByLogin(params.login)
  if (user.password == params.password) {
    session.user = user.login
    redirect(controller: «Event», action: «listEvent»)
  }
}
```

Si le résultat est bon, l'identification a bien lieu et on est renvoyé vers la liste des événements.
Dans l'exemple de filtre étudié ci-dessus, on a utilisé la possibilité du filtre de lancer son code avant l'action. Il est à noter qu'il existe trois moments où il peut s'exécuter :

- before : s'exécute avant l'action. S'il retourne false, cela indique que les filtres suivants et l'action ne seront pas traités.
- after: s'exécute après l'action. Il a la possibilité de modifier le rendu de la vue.
- afterView: s'exécute après le rendu de la vue.

Pour quitter l'application, ou simplement offrir la possibilité de changer d'utilisateur, une action de logout peut être implémentée en invalidant la session, puis en redirigeant vers la page de login (ou toutes autres pages, mais le filtre fera revenir à la page de login).

```
def logout() {
  session.invalidate()
  redirect(controller: «Event», action: «login»)
}
```

Suite des écrans

Il faut maintenant créer encore deux pages. La première pour la création d'un nouvel événement. Ici, aucune difficulté, une vue est un simple formulaire proposant deux champs, d'un côté... et de l'autre, une méthode associée dans le contrôleur *it.resto.EventController* qui se nommera *createEvent*.

```
if (params.name != null) {
    DateFormat df = new SimpleDateFormat(«yyyy-MM-dd'T'hh:mm»,
 Locale.FRENCH)
      def event = new Event(name: params.name, eventDate:df.
parse(params.dateEvent), owner:User.findByLogin(session.user)
).save()<
    redirect(controller: «Event», action: «listEvent»)
}
```

La vérification de l'existence du paramètre *name* passé en request, permet de tester que la formule a été envoyée.
Puis, l'événement est créé (avec nom, date et utilisateur en session) en une fois : *new* et *save*. Pour finir, le contrôleur se redirige vers la méthode *listEvent* du controleur *EventContoller*.
Dans ce cas, la vue suivra également et l'utilisateur arrivera sur la liste des événements en cours. Le nouvel événement est présent dans la liste. La seconde page présente en détails l'événement sélectionné et permet de voter au sein de celui-ci.
Pour cette page, il y a un peu plus de besoin fonctionnel:

- Afficher des Votes existants.
- Donner la possibilité de voter si ce n'est pas déjà fait par l'utilisateur.
- Tous ces traitements (création des votes et renvois de la liste des votes pour un événement spécifique) sont réalisés par un service *it.resto.EventService*.

Pour créer un service, on reste toujours dans la même philosophie pour Grails :

```
grails create-service it.resto.Event
```

Cette classe va contenir deux services.
Un premier pour créer des votes, prenant en compte la liste des restaurants sélectionnés.

```
def createNewVote(Event event, List restaurants, User user) {
    def vote = new Vote(user:user, event:event, restaurants:restaurants)
    event.addToVotes(vote)
    vote.save(failOnError: true)
    event.save(failOnError: true)
  }
```

Un vote est constitué d'un utilisateur, d'un événement et d'une liste de restaurants sélectionnés.
Dès la création du Vote, il est possible de passer en paramètre l'ensemble des éléments qui le composent (User, Event et liste de Restaurant).
Pour terminer, le vote doit être ajouté à l'événement Event avant que tout soit sauvegardé.
Le second service est l'équivalent en lecture. Il retourne une description des votes d'un événement, sous forme d'une liste de map, contenant les utilisateurs et les résultats de leurs votes.

```
List getVotesDesc(Event event) {
  def restaurants = Restaurant.listOrderByName()
  def votes = event.getVotes()

    def resultMap = []
    for (vote in votes) {
        def restoList = []
        for (restaurant in restaurants) {
          restoList.add(vote.getRestaurants().contains(restaurant))
        }
    def tmp = [
      user:vote.getUser(),
      listVotesByRestaurants:restoList
    ]
     resultMap.add(tmp)
     }
     return resultMap
  }
```

La méthode renvoie une liste de Map contenant l'utilisateur ayant réalisé le vote, ainsi qu'une liste de boolean indiquant si les restaurants ont été sélectionnés ou non.
En groovy, une liste se déclare comme ceci : *def restoList = []*
Une map, quant à elle, peut être définie de cette façon :

```
      def tmp = [
          user:vote.getUser(),
          listVotesByRestaurants:restoList
      ]
```

L'injection du service dans le contrôleur se fait par une simple déclaration: *def eventService*
Encore une fois, convention plutôt que configuration.
La structure est renvoyée dans la GSP via le contrôleur.
Dans la gsp, cette structure est traitée de la façon suivante.

```
<g:each var=»vote» in=»${votes}»>
  <tr><td>${vote.user.lastName} ${vote.user.name}</td>
    <g:each var=»restaurantSelected» in=»${vote.listVotesBy
Restaurants}»>
      <td><img  src=»/it-resto/images/${restaurantSelected}
.png» width=»20»></td>
</g:each>
</tr>
</g:each>
```

Un peu de clarté dans les uri

Les URI du site sont dépendantes du nom des contrôleurs. Simple pour le développeur, mais un peu moins lisible pour les utilisateurs.
Le fichier grails-app/conf/ UrlMappings.groovy permet de gérer facilement la cartographie des URI. Nous allons modifier l'accès à l'application, en modifiant la règle du / en remplaçant la règle par celle-ci :

```
«/»(controller: «Event», action: «listEvent»)
```

En arrivant dans l'application, on voit directement le rendu de la méthode listEvent d'EventController.groovy, et non plus la liste des contrôleurs disponibles.

Génération d'une interface d'administration avec le scaffolding

Toute l'application est maintenant prête à être utilisée, mais il n'existe, pour l'instant aucun moyen simple de l'administrer (créer/supprimer un utilisateur, ajouter des restaurants ou toutes autres tâches qui seront de la responsabilité de l'administrateur de l'application).
Afin de faire cela rapidement, Grails propose un mécanisme de scaffolding, qui génère les interfaces nécessaires pour intervenir en CRUD (Create, Read, Update et Delete) sur les différents objets du domain.
Il faut créer un contrôleur pour chaque domain.

```
grails create-controller it.resto.Vote
grails create-controller it.resto.User
grails create-controller it.resto.Restaurant
```

Les trois nouveaux contrôleurs sont créés ainsi que les classes de tests unitaires associées.
Dans chacun des quatre Controller existants, il faut ajouter la ligne suivante :

```
static scaffold = true
```

Voilà, l'interface d'administration est prête!
Si vous trouvez que les libellés ne sont pas très lisibles.
N'hésitez pas à ajouter une méthode *toString* dans chaque définition des objets du domain (voir Figure 2).
Exemple pour it.resto.User

```
String toString() {
  return this.lastName + ' ' + this.name
}
```

Petit inconvénient de la méthode, l'interface administrateur est accessible à tous les utilisateurs du service pour l'instant. Définissons les utilisateurs ayant les capacités d'un administrateur, en ajoutant une propriété *admin* dans l'objet domaim *it.resto.User*

```
boolean admin= false
```

Pour éviter tout problème, la valeur est initialisée à false (utilisateur sans droit administrateur).
Seconde étape, générer les GSP associées aux interfaces du scaffolding afin de les spécialiser en limitant leurs accès (de la même façon, sur d'autres projets, il est possible de modifier l'aspect graphique).

```
create-view it.resto.Vote
create-view it.resto.Event
create-view it.resto.User
create-view it.resto.Restaurant
```

Pour un Event, on retrouve les fichiers générés dans grails-app/views/event. On y trouve :

- _form.gsp: la forme qui sera utilisée pour la création et la modification d'un élément,
- create.gsp: page de génération d'un nouvel élément,
- edit.gsp: modification d'un élément,
- index.gsp: liste des éléments présents en base,
- show.gsp: détails d'un élément.

Il faut donc mettre en place une nouvelle taglib : *it.resto.tag. AdminTagLib* qui va se charger de filtrer en fonction des droits utilisateurs (s'agit-il d'un administrateur ou non ?). Encore une fois, il faut appeler les commandes Grails de création d'une taglib.
Cette fois, l'encodage (*defaultEncodeAs*) est défini à raw, afin d'éviter que les messages renvoyés soient transformés en HTML.

```
def isAdmin = { attrs, body ->
  if (session.userAdmin) {
    out << body()
```

Fig.2

Interface d'administration des utilisateurs

```
    }
    else {
      if (attrs.errorMsg != null) {
        out << attrs.errorMsg
      }
    }
  }
```

Ici, si l'utilisateur identifié dans l'application est administrateur, alors le contenu du tag est affiché, sinon, c'est le message d'erreur présent dans la taglib qui est renvoyé. La valeur de *userAdmin* en session est à ajouter dans l'action d'authentification. La taglib est ajoutée dans chaque pages générées (sauf _form.gsp), de la façon suivante :

```
<body>
  <g:isAdmin errorMsg=»Non autorisé»>
  (...)
  </g:idAdmin>
</body>
```

Cette fois, nous y sommes. Il ne manque qu'une page listant les quatre pages d'index de chaque domain. Une page admin.gsp avec les autres vues fait l'affaire. Pendant que nous y sommes, ajoutons également un lien dans les différentes pages de l'application, toujours en utilisant le même tag-lib, mais cette fois, sans message d'erreur.

```
<g:isAdmin>
  <a href=»admin» class=»btn btn-info btn-xs» type=»button»>>>
Administration</a>
</g:isAdmin>
```

Pour tester, relancer l'application : *run-app*

Plugins

Nous avons vu toutes les possibilités de Grails en termes de génération de classe, de test, de génération de méthode au runtime (via GORM). Mais, une des grandes forces du framework est le nombre de plugins disponibles (http://grails.org/plugins/) pour réaliser ces applications (actuellement, plus d'un millier) mais aussi la facilité d'utilisation dans un projet. Pour illustrer cela, nous allons ajouter la dernière fonctionnalité manquante dans l'application: permettre de supprimer les *it.resto.Event* déjà réalisés, c'est à dire ceux dont la date (*eventDate*) est dépassée dans le temps. Pour cela, on utilise le plugin Quartz 2.X Scheduler (http://grails.org/plugin/quartz2). L'ajout de plugin se fait simplement en ajoutant sa référence dans le fichier BuildConfig.groovy (grails-app/conf/) dans la propriété *grails.project.dependency.resolution.plugins*. Avant que le plugin soit pris en compte, il faut recompiler le projet : *grails compile.* Le plugin est automatiquement récupéré et le framework Grails le prend en compte.
Dans ce cas, cette prise en compte se verra directement, car la commande *create-job* est disponible.

```
grails create-job it.resto.clean
```

Le fichier *CleanJob.groovy* est créé dans grails-app/jobs/it/resto

```
class CleanJob {
  static triggers = {
    cron name:'cronTrigger', startDelay:10000, cronExpression:
  '0 0 6 * * ?'
  }

  def execute() {
    def listPastEvent = Event.findAllByEventDateLessThan(new Date())
    for (event in listPastEvent) {
      event.delete()
    }
  }
}
```

La propriété triggers indique le type et la fréquence de l'exécution de la tâche. Ici, le trigger sera effectif 10 secondes après le lancement de l'application (*startDelay:10000*) et s'exécutera tous les jours à 6H du matin (*cronExpression: '0 0 6 * * ?'*). La méthode *execute* sera lancée à chaque échéance du trigger. Ici, elle recherche tous les Event dont la date est antérieure à l'instant du lancement du batch et les supprime. Les votes associés à l'Event seront aussi supprimés grâce au lien d'appartenance (*belongs*) entre le Vote et l'Event (déclaré dans la classe *it.resto.Vote*).
Point d'attention, il faut toujours vérifier que la version du plugin utilisé est bien compatible avec la version de Grails utilisée.

Déploiement final

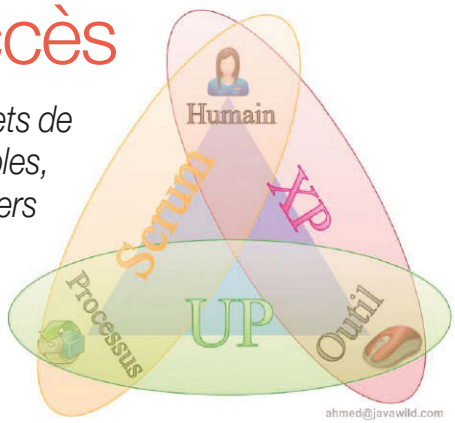
Le lancement de l'application a toujours été effectué avec la commande *run-app*, mais au moment de passer en production, il n'est pas recommandé d'utiliser cette méthode (uniquement pour le développement!).
En production, la documentation Grails insiste fortement sur la génération d'un war et le déploiement dans un conteneur de servlet compatible Servlet 2.5 (tel que Eclipse Virgo, GlassFish, IBM WebSphere, Jboss, Jetty, Oracle Weblogic, Resin, SpringSource tc Server ou Tomcat).
La liste est assez importante pour répondre à beaucoup de cas de configuration. Une documentation d'installation est également disponible sur le wiki Grails. La génération peut se faire en fonction de l'environnement d'exécution (pour rappel, voir la configuration des Data Sources qui inclut déjà les environnements de recette et de production).
Par exemple, pour la production, cela donnera : *grails prod war*

Conclusion

Grails est un framework souple et facile d'utilisation dans un cadre d'application Web sur une JVM. Tout au long de cet article, différents aspects du framework ont été abordés. De l'installation, à la création des différents éléments d'un projet pour finir par une mise en production potentielle. Mais, il ne s'agit que d'un aperçu et beaucoup de détails n'ont pas pu être couverts.
Pour aller plus loin, la documentation Grails est également très complete (http://grails.org/doc/2.3.1/guide/index.html). Les sources relatives à cet article sont sur GitHub : https://github.com/masson-r/it-resto

Rémi **Masson**
responsable de projet Web, Groupe Editis

L'agilité : un facteur clé de succès



L'agilité et ses pratiques changent la gestion et l'organisation des projets de développement informatique; elles les rendent plus réalistes et réalisables, tout en assurant un certain niveau de pragmatisme et de réactivité envers les demandes des clients et les besoins des utilisateurs. Ceci afin d'assurer la réalisation de produits d'une grande qualité sans dégrader la forte motivation des équipes impliquées dans les projets.

Les méthodes agiles ont vocation à apporter des solutions nouvelles à des problématiques de projets. Elles peuvent être appliquées dans tous les domaines, et toutes les structures peuvent y avoir recours.

L'agilité comme culture

Sur le marché, on trouve actuellement de nombreuses méthodes et processus agiles comme SCRUM, KANBAN, RUP, UP, XP, DSDM, SCRUMBAN, PUMA...etc. De nombreuses solutions Open Source ou payantes les implémentant existent. Ces méthodes agiles ont besoin de complémentarité entre elles; PUMA est considérée comme une approche plus complète que les autres mais difficile à implémenter.

L'agilité peut être considérée comme une culture évoluée de la gestion de projet ou du développement car elle permet une adaptation juste et ajustée aux changements d'environnement, en assurant l'efficacité et la flexibilité des parties concernées par l'approche agile. Elle peut être vue comme un mouvement révolutionnaire qui métamorphose de nombreuses structures et entreprises.

L'agilité et ses méthodes peuvent être qualifiées d'approches contemporaines et modernes au service de l'ingénierie logicielle et de la gestion de projet. Elles garantissent la qualité logicielle, la satisfaction client et une vie meilleure des équipes.

Aujourd'hui, de nombreux projets, des structures et des entreprises de toutes tailles migrent à l'agile dans la perspective d'accélérer, de sécuriser et de fiabiliser leurs processus internes et de pouvoir miser sur une rentabilité à grande valeur ajoutée, surtout en temps de crise économique ; être adaptatif, itératif et incrémental sont les piliers de l'approche agile. Il est plus facile de répondre à un nouveau besoin ou à un changement en mode agile qu'en mode classique dit traditionnel.

L'agilité limite les risques de la prédictibilité, de la sécurité et de l'anticipation qui caractérisent une démarche en cascade.

Un processus agile vise à augmenter l'implication directe du client et de l'utilisateur du produit final à livrer. Un tel processus se base sur un mode de communication dynamique et quasi quotidien, ce qui permet de faciliter les échanges entre les différents acteurs d'un projet ou d'une entreprise.

L'application de la méthodologie agile dépasse le cadre des projets de développement, et elle peut être adoptée pour différents types de structures et entreprises de tous les domaines (ressources humaines, journalisme, restauration...etc.).

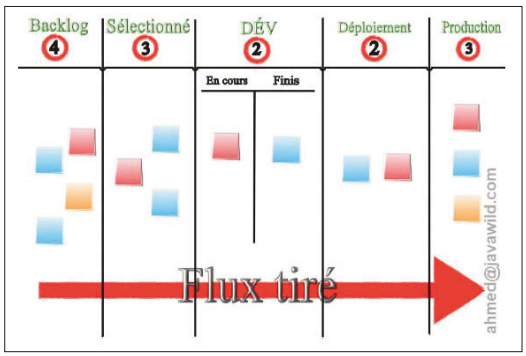
De nombreuses enquêtes sur les résultats de l'adoption de l'agilité par des entreprises de différents domaines, surtout dans l'innovation, ont révélé des changements positifs au niveau des coûts, de la satisfaction du client, de la qualité et de la productivité.

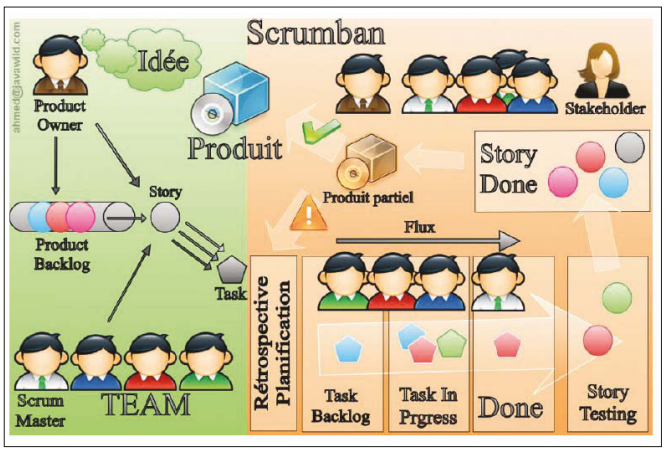
La résistance au changement reste le facteur humain principal de l'échec de la transition et de la mutation agiles. Toutefois la formation, l'accompagnement individuel ou collectif et le coaching sont l'expression de la volonté d'éviter tout échec lors de toute adoption de l'approche agile.

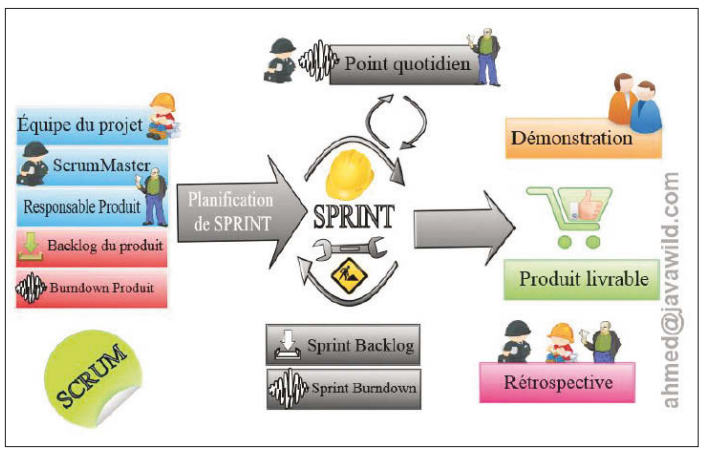
Toute transition vers l'agile nécessite de la formation et de l'accompagnement. Une transformation agile est une histoire d'hommes et de femmes, et pas seulement d'outils et de processus. La résistance humaine au changement demeure un obstacle majeur au succès d'une telle transformation. La structure, la nature et la culture des entreprises sont aussi des facteurs décisifs dans le processus de la migration agile.

Il faut bien signaler que la communication orale, la confiance, la polyvalence et l'initiative individuelle pour le bien collectif sont des comportements à encourager dans un cadre agile.

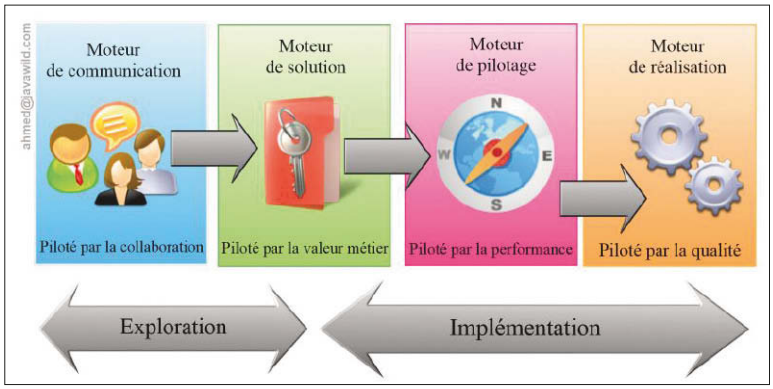
Actuellement, les méthodes agiles se diffusent à l'échelle mondiale et de nombreux groupements officiels et non officiels existent; ils orga-







nisent des évènements et des rencontres entre professionnels, experts, académiciens, chercheurs, adeptes et étudiants. Le mouvement agile n'est plus un courant d'amateurs, il s'organise dans le cadre d'organisations officielles à l'échelle mondiale et locale. L'alliance agile est l'une des organisations les plus connues du domaine. On trouve aussi d'autres organisations spécifiques comme celle de SCRUM (Scrum Alliance).



Le choix stratégique de passer à l'agile est justifié par la volonté de réduire les coûts, de raccourcir les délais, d'augmenter la productivité, d'impliquer directement le client ou l'utilisateur, de garantir une meilleure qualité du produit et d'offrir aux intervenants une qualité de vie d'entreprise soutenable et stable.

On peut citer l'exemple du Club Med qui a migré une partie de ses applications web vers des technologies moins coûteuses comme JAVA ou MYSQL selon une approche agile. Un autre exemple est celui de la British Airways qui a fait le choix d'exploiter de nouvelles ressources de revenus en réduisant les durées des développements et en augmentant la rapidité du retour sur investissement des systèmes livrés.

Le choix du service postal américain U.S Postal Service de migrer ses méthodes de développements du logiciel vers une approche agile a eu un effet très positif sur la rentabilité de l'entreprise et sur l'industrie postale aux États-Unis.

Succès et échecs

Un exemple de programme agile qui a tourné au fiasco est celui de l'Universal Credit du DWP (département du travail et des retraites du gouvernement britannique).

L'échec du programme de l'Universal Credit est lié à la taille du programme et aux bases de démarrage qui n'étaient pas du tout agiles, surtout au niveau de l'aspect contractuel qui a conclu un cadre en cascade pour une approche en agile. Un consultant qui a travaillé sur ce programme affirme que les intégrateurs implémentaient des pratiques en cascades présentées comme des méthodes agiles ; les développements se réalisaient d'une manière incrémentale sans être itératifs.

Ainsi, une formation de qualité, un accompagnement à moyen terme lors du processus de la transition agile, et un suivi à long terme durant la phase de la mutation agile sont nécessaires à l'accomplissement et à la réussite de l'adoption de l'approche agile pour un projet, une structure ou une entreprise.

L'agilité a démontré son efficacité au niveau de nombreux projets informatiques à l'échelle mondiale ; elle s'étend aujourd'hui à d'autres domaines et peut être déployée sur différents types de structures de toutes tailles.

De nombreux groupements, réseaux, alliances et associations ont vu le jour, ils organisent des évènements intéressants et incontournables autour du thème de l'agilité dans de nombreux pays.

Adam Ahmed **Ettarrouzi**
ahmed@javawild.com
Consultant JAVA J2EE (Agile).

La modélisation multidimensionnelle

Nous avons vu dans un précédent article, les tâches que devait mener un Data Scientist en entreprise. Parmi celles-ci, figure la modélisation des données. Un fondamental qui est à la base de tout système décisionnel. Si elle est mal faite, alors soyez sûr que vous serez embêtés tout le long de votre projet.

Avant d'aborder le sujet, levons quelques subtilités, incohérences et confusions qui se cachent sous la modélisation décisionnelle. Schéma en étoile, schéma en flocon, schéma en constellation, mesures, dimensions, OLTP, OLAP, Data Mart. Voilà autant de termes soumis à équivoque employés dans le jargon décisionnel. Pour tenter de lever ces confusions et doutes, commençons par le fondement : la base de données Entité/Association.

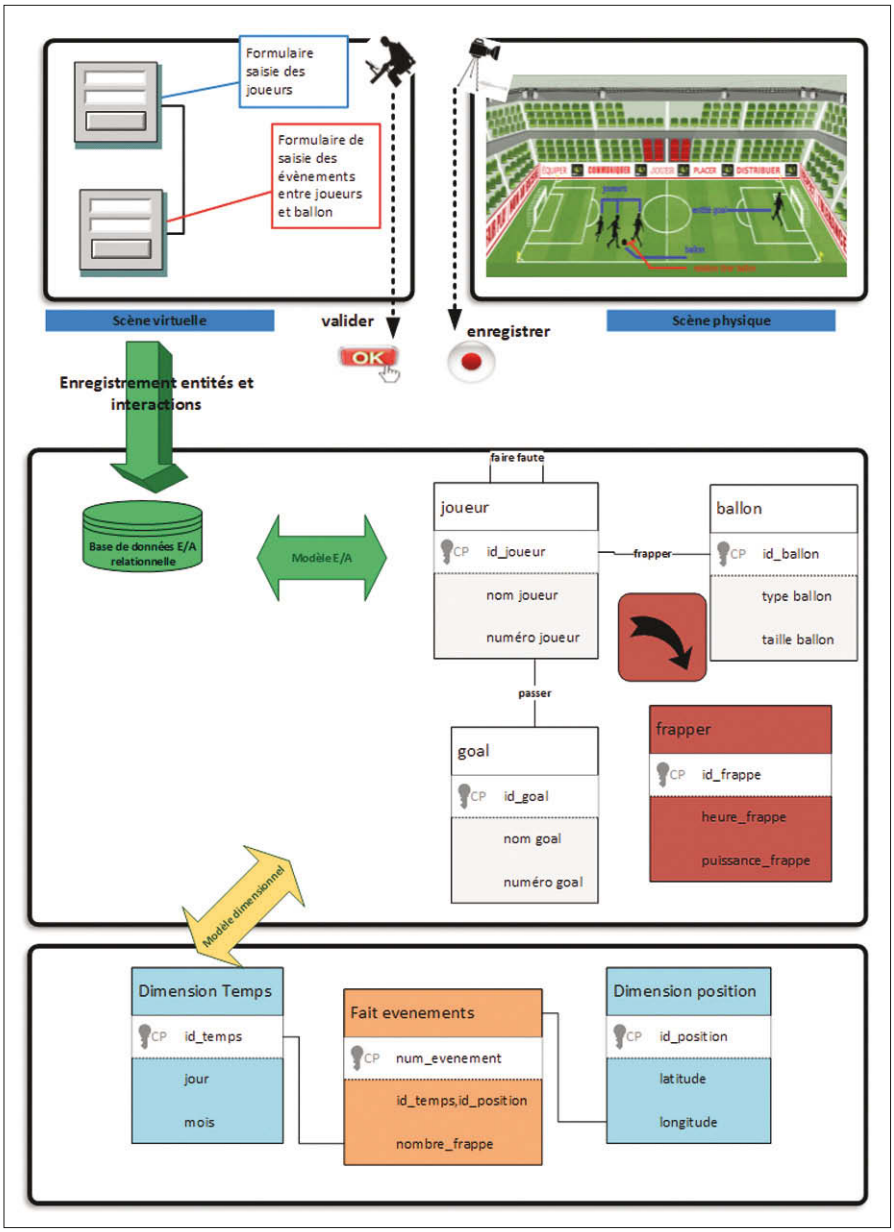
La représentation

Une base de données E/A classique, peut être décrite comme une scène filmée dans le temps, où se produisent des interactions entre divers acteurs. Cette représentation imagée va nous permettre de comprendre la différence discriminante qui existe entre une base de données qualifiée d'E/A et celle à connotation dimensionnelle.

En effet, alors que dans une base de données E/A, on se préoccupe d'enregistrer toute interaction entre deux ou plusieurs objets, d'où le nom de système OLTP (On Line Transactional Processing), dans une base de données décisionnelle, on s'intéresse plutôt à mesurer ces interactions à des fins d'analyses.

Pour être plus concret, prenons une scène de la vie réelle : vous assistez avec votre caméra à un match de football. Vous voulez faire 2 choses :

- Dans une première phase, vous voulez enregistrer toute interaction ayant lieu entre 2 ou plusieurs objets : une interaction entre joueur et ballon (un joueur frappe sur le ballon) ou encore une interaction entre deux joueurs (un joueur qui fait une faute sur un autre joueur).
- Dans une deuxième phase, vous voulez analyser les interactions que vous avez enregistrées dans la première phase : vous évaluez le nombre de coups francs directs tirés, le nombre de penalties ou encore le pourcentage de possession de ballon d'un joueur.



Dans la première phase, vous êtes en présence d'un système OLTP enregistrant les évènements ayant lieu sur la scène. Bien évidemment, dans la pratique, la scène se passe sur un jeu d'écrans composés de formulaires où les évènements sont représentés par des boutons de commande, qui, par clic, enregistrent les transactions dans les tables d'association. A ce stade, vous pouvez vous demander ce que représentent alors les acteurs ou objets de la scène. En fait, les acteurs sont représentés par les entités. Elles sont enregistrées dans les tables d'entités également appelées tables de référence ou encore tables de look up ou même dictionnaire dans certaines entreprises. L'ensemble Entités et Associations constituent le système OLTP.

Dans la deuxième phase, vous êtes en présence d'un système décisionnel, ou base de données OLAP (On Line Analytical Processing), où les enregistrements se font sous forme de métrique (nombre de coups francs par exemple) qui vont servir à analyser plus tard le modèle d'analyse ou de reporting. Mais en général, le système décisionnel n'est pas alimenté directement par enregistrement de la scène (c'est normal, puisqu'on ne peut pas s'amuser à faire l'analyse pendant qu'on enregistre le match !) mais en se basant plutôt sur le système OLTP sous-jacent. Cela est d'ailleurs réalisé par des outils d'intégration de données appelés outils ETL (Extraction Transformation et Chargement).

Maintenant que l'on connaît la réelle frontière

entre systèmes décisionnels et systèmes transactionnels (frontière qui n'est pas technique mais plutôt conceptuelle, car tous deux restent tout de même des bases de données relationnelles), nous pouvons commencer à voir les concepts au cœur de la modélisation : le Data Mart et les schémas en étoile, flocon et constellation. Le Data Warehouse est souvent ce que nous connaissons le plus des Systèmes d'Information d'Aide à la Décision (SIAD), alors qu'il n'est que l'union des Data Mart qui le constituent. Le Data Mart devient donc l'unité de base d'une modélisation. D'ailleurs j'entends généralement parler de Data Warehouse en étoile, ce qui en mon sens est rarement atteignable dans une entreprise. Par contre un Data Mart en étoile est bien plus réaliste (en général les Data Warehouses sont plutôt en constellation). Les Data Marts sont orientés sujet. En effet, pour modéliser un Data Mart, on s'intéresse tout d'abord à un sujet (par exemple, la performance d'une équipe). Ensuite de ce sujet, on extrait les métriques ou mesures ou indicateurs (par exemple nombre de coups francs, le pourcentage de possession de balle) participant à son étude ainsi que les axes d'analyse permettant d'évaluer ces mesures (le temps, la position etc.). Les dimensions ou axes d'analyse entourent les mesures ou indicateurs formant ainsi un schéma en étoile.

Par contre un schéma en étoile est une vision idyllique de la modélisation dimensionnelle préconisée par Ralph Kimball considéré comme le père de la modélisation dimensionnelle. Ainsi, bien que simple, performante et élégante, vous vous rendrez vite compte que pour le traitement de certains problèmes complexes tels que les structures hiérarchiques ou encore certaines structure N-N, vous êtes obligés de « floconner » votre schéma en étoile qui aura une structure en flocon. Pour plus d'information sur ces sujets, je vous suggère la bibliographie et la webographie kimballienne. Sur le sujet de la modélisation, le conseil est de toujours partir sur une structure en étoile pour ensuite le réajuster suivant le besoin en ayant recours au schéma en flocon de neige. Enfin, une fois le Data Mart modélisé, on obtient un Data Warehouse qui est une somme des Data Mart. C'est la raison pour laquelle le Data Warehouse ne peut généralement pas être en étoile puisqu'il représente une union de Data Mart. Il est donc en constellation.

En règle général, voici donc quelques étapes à suivre pour bien mener un projet de modélisation :

- Sélectionner le sujet à analyser (Par exemple la performance d'une équipe),
- Déclarer les indicateurs du sujet sélectionnés (pourcentage de possession, nombre de coups francs),
- Déclarer les dimensions qui vous permettrons d'analyser les indicateurs (le temps, le lieu de localisation de la balle etc..),
- Définir des hiérarchies qui permettront des analyses ascendantes ou descendantes (par exemple analyser le nombre de coup francs à la minute et à l'heure),
- Passer au sujet suivant à analyser.

Bien évidemment, certaines étapes telles que la définition de la granularité des faits qui décrit la manière dont vous capturez vos mesures (par exemple, vous pouvez capturer vos nombres de coups francs à la demi-seconde. Je sais que c'est un peu exagéré sauf pour le Barça!), mais cette omission est volontaire, pour vous inciter à la bibliographie pour un approfondissement du sujet. Vous vous êtes certainement aperçu que le processus de cette modélisation orienté sujet est itératif. Cela ouvre bien des perspectives. En effet, avec les méthodologies agiles, on peut modéliser le Data Warehouse de façon incrémentale, itération par itération. Cela n'était pas le cas avant, où la tendance était inmonienne (issue de Bill Inmon, un autre acteur désigné comme le père du Data Warehouse). La vision de Bill Inmon, qualifiée d'approche Top Down, consistait à partir d'un Data Warehouse Central pour le casser en multiple Data Marts. Dans ces circonstances, il fût assez difficile d'appliquer une méthode incrémentale.

Un peu de No SQL

Je termine cet article en parlant de NoSQL, considéré comme l'alternative aux bases Relationnelles et comme solutions aux Big Data. Je pense là aussi, qu'il est important d'éclairer le fait que les bases NoSQL ne peuvent remplacer les systèmes E/A. Ils ne sont là que pour offrir une alternative aux bases de données que nous connaissons jusque-là et qui mettent une couche abstraite (le modèle relationnel) entre les programmes et les données. La communauté NoSQL veut revenir en arrière en réconciliant l'IHM et le stockage mais sous une façon plus optimale. Cependant au niveau de la modélisation, on ne peut changer la vision OLAP, car il s'agit d'un modèle de conception et non d'un modèle de structuration du SGBD sous-jacent. Ainsi donc la présence d'un SGBD NoSQL ne dispense pas d'une modélisation orientée décisionnel.

Mohamed Taslimanka **Sylla**
Architecte Business Intelligence
SFEIR

Java 8 : La révolution des Lambdas expressions

1ere partie

Sorti officiellement le 18 mars dernier, Java 8 tend à s'inscrire comme une évolution majeure du langage au même titre que Java 5 en son temps. L'arrivée des Lambdas, avec l'ouverture à la programmation fonctionnelle qui en découle, constitue la raison principale d'une évolution qui s'annonce comme une révolution pour les développeurs. Dans cet article, nous nous proposons de faire un tour d'horizon des Lambdas expressions et des nouveautés associées.

Reportée à plusieurs reprises, la nouvelle mouture de Java est finalement sortie en Mars 2014. Ce serait un euphémisme de dire que cette version 8 était attendue de pied ferme depuis de longs mois par les développeurs de la communauté Java. L'introduction des Lambdas expressions au sein du langage étant bien entendu la principale raison de l'attente entourant cette sortie. Il faut dire que les possibilités introduites par les Lambdas vont radicalement changer la manière de développer des applications Java. Néanmoins, comme toute révolution technique, la clé du succès réside dans l'adoption par les utilisateurs qui, en l'occurrence ici, sont les développeurs. En effet, hors du cadre de certains enthousiastes de la plateforme, qu'en sera-t-il de l'adoption des Lambdas par la grande majorité des développeurs Java en entreprise alors même que Java 7 est encore un lointain rêve pour certains ! Pour accélérer l'adoption des Lambdas en mettant en avant leur intérêt, la clé va résider dans la mise à disposition d'articles, tutoriaux ou supports favorisant leur prise en main par les développeurs. C'est clairement dans cette démarche que s'inscrit cet article.

Pourquoi les Lambdas ?

Créé au milieu des années 90, le langage Java a tiré pleinement parti de l'essor des langages orientés objets en s'imposant rapidement comme le langage de référence en entreprise dans un monde où le paradigme objet était la référence pour le développement logiciel. Néanmoins, bien avant l'apparition du modèle objet, les langages fonctionnels tels que Lisp ou Scheme, existaient déjà, mais leur paradigme était restreint au cadre académique. Récemment, la programmation fonctionnelle est revenue en haut de l'affiche du fait de caractéristiques se prêtant particulièrement bien à la programmation concurrente et aux architectures réactives à base de bus d'évènements amenées a jouer un rôle crucial dès aujourd'hui en entreprise. Alors que penser ? Que le modèle objet a vécu et qu'il doit être supplanté par le paradigme fonctionnel ? La réponse n'est évidemment pas si tranchée et en pratique, la meilleure solution consiste à combiner au sein d'un même langage les 2 approches ouvrant de fait la voie à un certain nombre de possibilités du monde fonctionnel dans les langages objets. La plupart des langages majeurs, de Javascript à Python, en passant par C# et même le C++, ont adopté cette approche et introduit les Lambdas expressions. Plus étonnant encore, certains des langages phares tournant sur la JVM comme Groovy et Scala proposent déjà un tel support.

Conscients du problème, de nombreux développeurs avaient déjà remonté ce besoin pour Java dès 2006, soit peu de temps après la sortie de Java 5 ! L'inertie du monde Java étant ce qu'elle est, l'introduction des Lambdas s'est donc vue ralentie suite à d'innombrables discussions. Le tournant aura finalement été le rachat de Sun par Oracle en 2010 ainsi que la montée en puissance du slogan «Java est le nouveau Cobol». Souhaitant faire de Java un outil stratégique pour son avenir, Oracle a donc pris les choses en main et après encore bien des palabres et reports, nous voila arrivés en 2014 avec l'introduction des Lambdas au sein du JDK 8. Celle-ci devant permettre au langage de rester au coeur des applications d'entreprise pour de nombreuses années.

Qu'est-ce qu'une Lambda expression ?

Raccourci de Lambda expression, le terme Lambda vient de Lambda calcul, qui est un cadre théorique inventé par Alonzo Church dans les années 30. Ce système formel fonde les concepts de fonction et d'application. Ce petit rappel historique mis à part, on considèrera qu'une Lambda expression est un bloc de code, implémentant une fonctionnalité, pouvant être passée en paramètre pour une exécution future. Fondamentalement, il s'agit simplement d'une façon plus rapide pour implémenter des blocs de code à exécution différée.

Avant les Lambdas ?

Le spectre des besoins couverts par les Lambdas est vaste allant de la programmation concurrente aux kits graphiques tels que Swing. Ces besoins concernent l'exécution différée de blocs de code. De tels blocs étant ainsi utilisés pour répondre aux évènements utilisateurs sur une IHM. La solution palliant à l'absence de Lambdas jusqu'à présent fut de considérer que ces blocs de code étaient des objets et Sun mit alors à disposition un ensemble d'interfaces «listener» dont chacune couvrait des besoins spécifiques. Pour rendre l'implémentation de ces interfaces plus facile à écrire, Sun proposa ensuite le concept d'inner classes qu'il était possible de définir de manière anonyme. Les exemples de ces interfaces sont également légions hors du domaine IHM, et l'on pense tout de suite aux interfaces Runnable et Comparator. Prenons ainsi l'exemple d'un traitement que l'on souhaiterait exécuter dans un thread séparé. La solution jusqu'alors consistait à écrire le code suivant :

```
new Thread(new Runnable() {
 public void run() {
  doWork();
 }
).start();
```

Ici, l'inner classe utilisée est anonyme, mais on aurait très bien pu la référencer via une variable. Dans le même ordre d'idée, nous pouvons considérer le tri d'une liste de strings à l'aide d'un Comparator :

```
class LengthComparator implements Comparator<String> {
 public int compare(String first, String second) {
  return Integer.compare(first.length(), second.length());
  }
}

LengthComparator lengthComparator = new LengthComparator();
Arrays.sort(strings, lengthComparator);
```

Enfin, pour revenir à notre exemple concernant les interfaces graphiques, on pourrait proposer le code JavaFX suivant pour réagir au clic sur un bouton :

```
button.setOnAction(new EventHandler<ActionEvent>() {
 public void handle(ActionEvent event) {
  System.out.println(«Click»);
 }
});
```

Ces exemples de code mettent en avant la même approche : un bloc de code passé sous la forme d'un objet en entrée de méthode, le tout avec une exécution différée. Comme vu précédemment, il est possible dans d'autres langages d'utiliser de tels blocs sans passer par de l'objet.
Du côté des concepteurs du langage Java, la résistance fut grande avant d'ajouter une telle fonctionnalité du fait d'une volonté de garder le langage aussi simple et consistant que possible.
Cependant, comme nous le verrons par la suite, les Lambdas ne servent pas seulement à raccourcir la syntaxe d'écriture des blocs de code, mais permettent de concevoir des APIs plus simples, plus consistantes et surtout plus puissantes. Les nombreuses années de discussions autour des Lambdas auront finalement fait émerger une solution puissante et particulièrement bien adaptée à Java comme nous allons le voir à présent.

Syntaxe

Au niveau du langage Java, la syntaxe retenue consiste en 3 parties :

- Un ensemble de paramètres entourés par des parenthèses lorsqu'il y a plus d'un paramètre
- Une flèche ->
- Un corps qui peut être une expression simple ou un bloc de code Java

En reprenant l'exemple précédent du Comparator présenté sous la forme d'une inner classe anonyme, Java 8 permet l'équivalence suivante avec une Lambda expression :

```
Arrays.sort(strings, (String first, String second) -> Integer.
compare(first.length(), second.length());
```

Cette première expression Lambda montre clairement l'intérêt d'une telle syntaxe. Ici, une seule expression constitue le corps de la Lambda mais il est possible d'écrire un bloc complet :

```
Arrays.sort(strings, (String first, String second) ->
{
 System.out.println(«Comparaison « + first + « avec « + second);
  return Integer.compare(first.length(), second.length();
});
```

Si le corps d'une Lambda s'approche de celui d'une méthode, il y a en revanche certaines limitations relevant du bon sens. Ainsi, le corps d'une Lambda ne peut faire de break ou de continue en dehors de l'expression, et si une Lambda retourne une valeur, chaque chemin de code doit retourner une valeur, idem pour les exceptions.
Renforcée avec Java 8, l'inférence de type évite au développeur de devoir spécifier le type de retour d'une Lambda. Celui-ci est toujours inféré depuis le contexte. En outre, cette inférence peut également s'appliquer sur les paramètres d'entrée dans un certain nombre de cas, ce qui peut donner :

```
Comparator<String> lengthComp = (first, second) -> Integer.
compare(first.length(), second.length());
```

Interfaces fonctionnelles

Les interfaces ne possédant qu'une seule méthode, créées principalement pour répondre au besoin d'encapsulation de blocs de code à exécution différée, bénéficient naturellement des Lambdas expressions comme nous avons pu le remarquer. Ainsi, toute interface ne possédant qu'une seule méthode abstraite peut être utilisée comme une Lambda. Ce type d'interface est désormais appelée interface fonctionnelle. Mais les méthodes d'une interface ne sont-elles pas par nature toutes abstraites ? Avant oui, mais avec Java 8, la donne change avec l'introduction des méthodes par défaut dans les interfaces, ce qui permet de définir des méthodes non abstraites au sein des interfacesn comme nous le verrons par la suite. Techniquement parlant, en reprenant l'exemple précédent, la méthode de tri Arrays.sort reçoit un objet d'une classe implémentant Comparator<String> et invoque ensuite la méthode compare de cet objet qui exécute le corps de la Lambda expression. La gestion de ces objets est transparente pour le développeur, et se révèle bien plus performante que les inner classes. En pratique, il est sûrement plus simple de considérer les Lambdas comme des fonctions et non des objets, et d'accepter le fait qu'elles puissent être passées comme des interfaces fonctionnelles. L'exemple du clic sur un bouton devient tout à coup particulièrement simple :

```
button.setOnAction(event -> System.out.println(«Click»));
```

En réalité, la conversion d'une expression Lambda ne se fait que vers une interface fonctionnelle. Là où d'autres langages autorisent la définition et l'emploi d'un type de fonction tel que (String, String) -> int, Java ne permet même pas d'assigner une Lambda à un type Object ! Pour la première fois dans l'histoire de Java, nous trouvons ainsi quelque chose qui ne peut être assigné à une référence de type Object. En effet, Object n'étant pas une interface fonctionnelle, la ligne de code suivante ne compilera pas :

```
Object o = () -> System.out.println(«Hello»);
```

Ici, le compilateur ne peut savoir à quelle interface fonctionnelle il doit rattacher la Lambda expression. L'astuce consiste à s'appuyer sur l'inférence de type en aidant le compilateur de la sorte :

```
Object o = (Runnable) () -> System.out.println(«Hello»);
```

Evidemment, le développeur peut créer suivant ses besoins ses propres interfaces fonctionnelles utilisables sous forme de Lambdas. Afin de permettre au compilateur de vous avertir de la déclaration de plus d'une méthode abstraite au sein d'une interface devant être fonctionnelle, l'annotation @FunctionalInterface est proposée. Optionnelle, elle offre, en plus de la vérification à la compilation, le marquage en tant que tel qu'au sein de la Javadoc, ce qui s'avère pratique dans l'effort de documentation. Un dernier mot concernant les exceptions checked qui peuvent poser problème lors de la conversion d'une Lambda en instance d'interface fonctionnelle. Il faut noter que si la méthode abstraite de l'interface fonctionnelle ne déclare pas que la méthode peut lancer une exception, l'utilisation d'une méthode pouvant lancer cette exception dans le corps d'une Lambda est interdite et provoquera une erreur de compilation :

```
Runnable sleeper = () -> { System.out.println(«Zzz»); Thread.
sleep(1000); };
// Erreur de compilation : Thread.sleep peut lancer une
checkedInterruptedException
```

La méthode run de Runnable ne déclarant pas pouvoir lancer d'exception, il est nécessaire soit de cacher l'exception dans le corps de la Lambda,

soit de passer par une interface fonctionnelle déclarant une telle exception. Dans le cas présent, l'interface Callable<Void> ferait parfaitement l'affaire.

Références de méthodes

Quelquefois, il se peut qu'une méthode existante effectue déjà le travail que l'on souhaite réaliser dans une Lambda. Prenons pour exemple l'utilisation de la méthode setOnAction d'un bouton déjà présenté plus haut :

```
button.setOnAction(event -> System.out.println(event));
```

Pour adresser ce type de besoin, les références de méthodes ont été ajoutées à Java 8 ce qui donne l'équivalence suivante :

```
button.setOnAction(System.out::println);
```

La référence de méthode System.out::println est équivalente à la Lambda expression suivante :

```
x -> System.out.println(x)
```

Le spectre d'emploi de ces références de méthodes est large et le développeur y trouvera vite son compte. Elles peuvent être utilisées dans 3 contextes :

```
object::instanceMethod
Class::staticMethod
Class::instanceMethod
```

Les 2 premiers cas sont équivalents à l'exemple utilisé sur le bouton JavaFX. Ainsi, l'utilisation de la méthode statique pow de la classe Math sous la forme Math::pow est équivalente à cette Lambda :

```
(x,y) -> Math.pow(x,y)
```

Le dernier cas diffère quelque peu puisque le premier paramètre devient la cible d'appel de la méthode référencée. Prenons l'exemple du tri d'une liste de strings à l'aide de la méthode compareToIgnoreCase de la classe String :

```
Arrays.sort(strings, String::compareToIgnoreCase);
```

La reference String::compareToIgnoreCase utilisée ici sera traduite en Lambda expression en utilisant le premier paramètre comme cible pour l'appel de méthode :

```
(x,y) -> x.compareToIgnoreCase(y)
```

En outre, rajoutons que l'inférence de type s'applique bien évidemment sur les références de méthodes comme cela est le cas habituellement en Java. Enfin, il est possible de capturer le paramètre this dans une instance de classe ou bien encore le paramètre super. La référence this::equals sera ainsi équivalente à la Lambda suivante :

```
x -> this.equals(x)
```

Références de constructeurs

Sur le même principe que les références de méthodes, les références de constructeurs font leur apparition. Ainsi, Button::new est une référence au constructeur de la classe Button. Le constructeur à appeler va dépendre du contexte dans lequel la référence est employée. Considérons l'exemple suivant :

```
List<String> labels = new ArrayList<String>();
// …
Stream<Button> stream = labels.stream().map(Button::new);
List<Button> buttons = stream.collect(Collectors.toList());
```

Ici, le contexte entourant l'emploi de la référence au constructeur de Button permet au compilateur d'inférer le constructeur à appeler en l'occurrence Button(String). Précisons que les détails sur les nouveautés présentes dans cette portion de code telles que les Streams seront largement détaillées dans la seconde partie de cet article.
Les possibilités offertes par ces constructeurs de références sont nombreuses et l'on peut, par exemple, les utiliser avec un tableau. Par conséquent, int[]::new est une référence valide transformée en Lambda expression de telle sorte que le paramètre en entrée soit la taille du tableau à créer :

```
x -> new int[x]
```

A première vue, l'utilité de cette syntaxe ne saute pas aux yeux. Néanmoins, les concepteurs d'API en tireront rapidement profit pour palier une limitation du langage Java. En effet, il est impossible de créer un tableau d'un type générique T donné. L'expression new T[n] provoquant une erreur de compilation. L'utilisation conjointe de la référence d'un constructeur avec les fonctionnalités de l'API Stream ajoutée à Java 8 ouvrent désormais la voie à une construction de la forme suivante :

```
Button[] buttons = stream.toArray(Button[]::new);
```

Scope des variables

Au niveau du scope des variables, les Lambdas reconnaissent l'environnement immédiat avant leur définition comme le scope de niveau extérieur le plus proche. Prenons ainsi l'exemple suivant :

```
class Hello {
  public Runnable r = () -> {
  System.out.println(this);
  System.out.println(toString());
 };

 public String toString() {
  return «Hello depuis toString()»;
 }
}
```

Le paramètre this et la méthode toString() font ici référence aux objets de la classe Hello, contrairement à ce qui se serait passé avec une inner classe où ces paramètres auraient fait référence à la classe Runnable nouvellement créée.
Les inner classes ne peuvent référencer que des variables de type final. Les Lambdas relâchent quelque peu cette contrainte puisqu'il est possible maintenant de référencer des variables dites libres tant qu'elles sont effectivement finales même si elles ne sont pas définies avec le mot clé final. De fait, le code suivant est autorisé :

```
public static void main(String... args) {
 String message = «Hello»;
 Runnable r = () -> System.out.println(message);
```

```
  r.run();
}
```

La référence à l'objet message n'étant jamais modifiée à l'intérieur du scope de la méthode main, l'objet est effectivement final et peut être capturé par la Lambda. En effet, les Lambdas expressions proposées par Java 8 doivent stocker les valeurs de ce type de variables libres. On dit alors que ces valeurs ont été capturées par l'expression Lambda ce qui démontre de fait que ces Lambdas sont des closures. Précisons également que les inner classes l'étaient aussi, mais que Java 8 nous propose une syntaxe beaucoup plus attractive avec les Lambdas. Une fois la notion de capture de valeurs assimilée, on comprend pourquoi le code suivant produit une erreur de compilation :

```
public static void repeatMessage(String text, int count) {
 Runnable r = () -> {
  while (count > 0) {
   count--; // Erreur de compilation
   System.out.println(text);
   Thread.yield();
   }
 };
  new Thread(r).start();
}
```

Ici, la variable libre count ne peut être modifiée dans la Lambda. La raison de cette interdiction tient au fait que modifier une variable au sein d'une Lambda ne serait pas thread-safe. Enfin, s'il est possible de modifier un objet passé au sein d'une Lambda, du moment qu'on ne touche pas à sa référence, ce type de code est à proscrire du fait qu'il n'est pas thread-safe :

```
List<Path> matches = new ArrayList<>();
for (Path p : files)
  new Thread(() -> { if (p … ) matches.add(p); }).start(); // A proscrire
```

Méthodes par défaut

Un des principaux reproches faits aux interfaces jusqu'alors vient du fait qu'elles ne peuvent proposer d'implémentations par défaut, même quand celle-ci paraît totalement évidente. Attardons nous sur la classe Iterator générique ci-dessous :

```
interface Iterator<T> {
  boolean hasNext();
  T next();
  void remove();
}
```

Etant donné cette définition, il apparaît clairement qu'il est possible de définir une méthode skip, permettant de sauter jusqu'au prochain objet, à partir des méthodes précédentes. De fait, une implémentation par défaut serait tout à fait pertinente en lieu et place de l'utilisation d'une classe abstraite. Avec les méthodes virtuelles d'extension, également connues sous le nom de méthodes par défaut, Java 8 permet désormais aux interfaces de proposer une implémentation par défaut pour une méthode :

```
interface Iterator<T> {
   // …
  void skip(int i) default {
    for (; i > 0 && hasNext(); i--) next();
  }
}
```

Cet ajout au langage s'avérait également nécessaire pour mettre à jour certaines API de Java telles que l'API Collections, en proposant de nouvelles fonctionnalités tirant parti des Lambdas, sans obliger tous les programmes l'utilisant, à implémenter les nouvelles méthodes des interfaces. Ainsi, ils ont pu ajouter la méthode forEach à l'interface Iterable sans que cela ne vienne impacter tous les programmes existants.

Au sein de la communauté Java, certains puristes sont tout de suite montés au créneau pour dénoncer cette évolution arguant du fait qu'il s'agissait d'un mécanisme visant à palier le faible pouvoir déclaratif des interfaces, tout en créant un cadre d'utilisation autorisant l'héritage multiple en Java. Si cela est en partie vrai, il convient d'être pragmatique et de considérer ces méthodes d'extension virtuelles à leur juste valeur. Il s'agit avant tout d'un mécanisme puissant mis à disposition des développeurs pour étendre des interfaces existantes en se rapprochant par exemple des traits de Scala. Les APIs du JDK étant les premières à en bénéficier puisque certaines ont subi un sérieux lifting.

Enfin, les problèmes potentiels d'héritages multiples induits par ces méthodes virtuelles d'extensions ne sont qu'un faible prix à payer au regard des gains apportés.

A charge donc aux développeurs de prendre garde à ces problèmes, en gardant bien à l'esprit les règles de priorité s'appliquant dans des situations d'héritage multiple avec Java 8.

Méthodes statiques et interfaces

Autre nouveauté apportée par Java 8 côté interfaces, la possibilité d'y définir des méthodes statiques. Aucune raison technique ne s'y opposait et seules des contre-indications au niveau de l'esprit de l'abstraction des interfaces étaient mises en avant. Au niveau JDK, cette autorisation profite par exemple à des classes comme Collections ou Paths. La méthode Paths.get(«jdk1.8.0»,»jre»,»bin») qui renvoie un Path peut ainsi être implémentée au sein d'une méthode statique dans l'interface Path :

```
public interface Path {
     public static Path get(String first, String... more) {
       return FileSystems.getDefault().getPath(first, more);
      }
      ...
  }
```

D'autres exemples peuvent être remontés du JDK pour cette nouveauté, qui, si elle ne révolutionnera pas le langage, apporte un outil supplémentaire aux développeurs et aux concepteurs d'APIs.

Conclusion

Attendues depuis de nombreuses années maintenant, les Lambdas expressions vont apporter un changement important dans l'univers Java, tant du point de vue du code produit, que de la conception des APIs. Les possibilités de programmation fonctionnelle qu'elles introduisent constituant à la fois une opportunité et un enjeu de taille auquel les développeurs devront faire face. La première partie de cet article aura permis de présenter les Lambdas en profondeur.

La seconde partie permettra quant à elle de rentrer de manière plus pratique dans les Lambdas avec la mise en avant de leurs utilisations au quotidien. Cet usage nous permettra également d'introduire une autre grande nouveauté de Java 8, à savoir les Streams ainsi que les nouveautés que cela induit au niveau de certaines APIs telles que l'API Collections notamment.

Sylvain **Saurel** – *Ingénieur d'Etudes Java / Java EE*
sylvain.saurel@gmail.com

Prototyper son application

Dans cet article, je vais vous exposer ma méthode de travail de designer sur un projet d'application. J'espère vous fournir certaines clefs pour bien mener et réussir vos applications tant d'un point de vue ergonomique que graphique.

Le maître mot lors de la création d'une app, qu'elle soit mobile ou tablette, est la **cohérence**.

- Cohérence dans l'arborescence de votre projet, vos différentes pages de contenu doivent être organisées de façon logique. Cette logique doit être comprise en un clin d'œil par l'utilisateur; il sera donc primordial de prendre en compte le propos de l'application et le public visé.
- Cohérence dans la navigation, l'utilisateur doit s'y retrouver immédiatement et ses choix de navigation doivent l'amener aux résultats souhaités rapidement et de façon fluide. Une grande part de l'expérience utilisateur se jouera ici.
- Cohérence graphique bien sûr, qui implique de définir des codes et une charte qui sera appliquée de manière égale à l'ensemble de vos pages.

Pour assurer cette cohérence globale, il faut éviter de commencer à travailler avant d'avoir en main tous les éléments et informations concernant le projet. L'ajout à la dernière minute d'une nouvelle page ou d'un nouvel onglet pourrait casser votre structure et vous fera perdre un temps précieux en réorganisation. C'est pourquoi avant de commencer le prototypage assurez-vous d'avoir une bonne vision globale du projet.

Note : *tout au long de cet article, je m'appuierai sur mon récent projet de refonte graphique de l'application Collecto pour Windows Phone, développée par David Catuhe.*

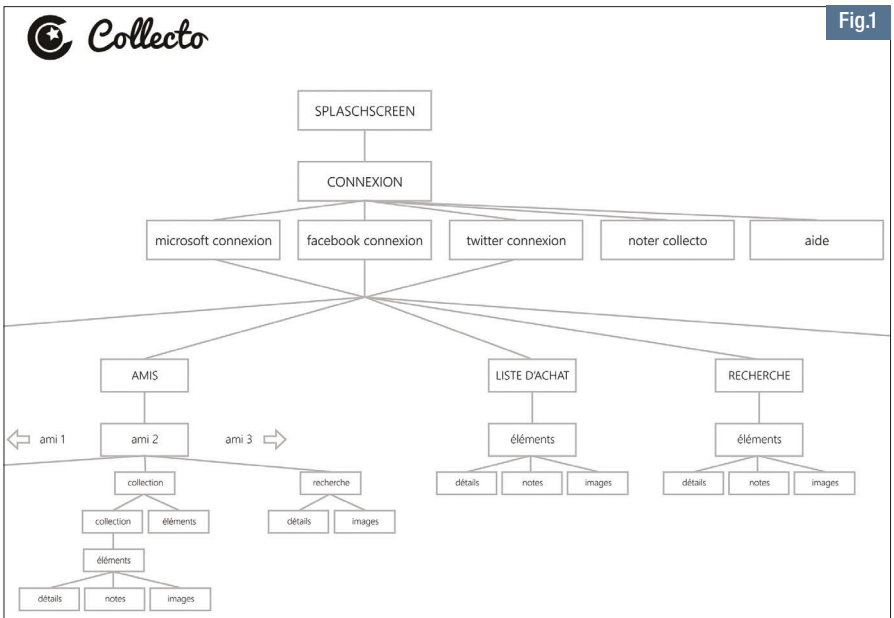
L'arborescence

Établir une arborescence complète de votre future application peut être fastidieux, mais il est indispensable si vous souhaitez avoir du recul sur le projet et faire de bons choix organisationnels. Votre architecture de base dépendra en grande partie de la nature même de votre app. Vous aurez ainsi le choix entre regrouper votre contenu par cible d'utilisateur (quel est le public visé ?), par tâche (exemple : acheter/ vendre), par type de contenu, par zones géographiques, etc.

Ces premiers grands ensembles de contenu définissent vos rubriques. Il vous faudra ensuite définir vos sous-rubriques et ainsi de suite jusqu'à classer tous les contenus disponibles sur votre application Fig.1.

Gardez à l'esprit qu'une application n'est pas un site web et se construit généralement autour d'une ou deux fonctionnalités principales. Les informations ou les fonctions doivent être accessibles quasi immédiatement. Évitez donc une arborescence trop complexe.

Fig.1



Arborescence partielle de l'application Collecto.

Certains outils simples existent qui permettent de créer des arborescences par exemple Microsoft Visio, Word, ou encore certains outils en ligne comme www.draw.io

Le mockup

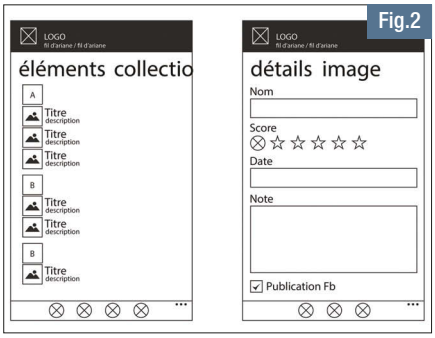
Le *mockup* n'est pas une maquette, il ne sert en aucun cas à « habiller » l'application, mais bien à définir les zones de contenu (zone de texte, images, etc.), l'emplacement des différents éléments (boutons, progress bar, icônes, etc.). Dans le web on appelle cela le *zoning* ou encore *wireframing* (structure fil de fer).

Si votre étape d'arborescence a été bien réalisée, vous constaterez qu'il sera plus simple de prendre chaque groupe de contenu, défini au préalable, pour les poser sur la page, et ainsi leur définir un emplacement spécifique en respectant leur importance hiérarchique. Le mockup est donc la première ébauche d'une mise en page, et, même si aucun style n'est vraiment défini, chaque élément posé sur la page doit être identifié pour ce qu'il est. On doit donc être en mesure de distinguer les zones de texte des zones d'image, les vidéos, les différents types de bouton, etc. Fig.2.

Il existe de nombreux outils qui permettent de créer des mockups. Adobe Fireworks en est un très bon qui contient un grand nombre de templates mobiles de toutes marques (plus de templates sont téléchargeables gratuitement sur le web). Via un simple drag & drop on peut choisir des éléments dans une liste assez exhaustive pour les placer sur la page et ainsi ébaucher rapidement un premier concept. La force de Fireworks est de permettre d'ajouter des interactions sur les éléments de la page et ainsi de proposer à son client un mockup fonctionnel qui permettra de naviguer d'une page à l'autre. Seul bémol de Fireworks, même s'il est toujours supporté et disponible pour le moment, Adobe a stoppé les développements.

Toujours chez Adobe, le logiciel de dessin vectoriel Illustrator, couplé à des templates d'interface, est une alternative puissante même s'il ne propose pas d'interactivité. Des sites comme Webalys proposent bon nombre d'éléments vectoriels, templates et icônes qui permettent

Fig.2



Mockups de deux écrans de l'application Collecto

de gagner en temps et en efficacité. Chez Microsoft on trouve l'équivalent de Fireworks avec Blend qui propose les mêmes fonctionnalités, interactivité comprise. Blend en version light est gratuit et proposé en téléchargement avec Visual Studio.

Balsamiq est un outil intéressant qu'il faut évoquer ici, car il propose un grand nombre de templates dont des templates Windows Phone ce qui est pour l'instant assez rare pour être noté.

Enfin il existe de nombreux outils gratuits en ligne qui permettent de faire des choses très propres par exemple Wirify, Moqups ou encore Mocking Bird.

La charte graphique

Imaginons que je vous propose deux jeux vidéo au concept identique : préférerez-vous jouer au jeu bien fait, mais sans originalité graphique, ou à celui qui vous offre une expérience visuelle inédite ? (question purement rhétorique évidemment). Pour votre application le constat est le même. Que votre projet soit basé sur une idée novatrice ou qu'il s'agisse d'un portage, se distinguer de la masse par le style c'est attirer l'attention sur votre produit et donc générer plus de téléchargement.

Avant de vous jeter dans l'arène en étalant de la peinture partout sur vos beaux *mockups* tout propres je vous propose de passer par une étape intermédiaire, la *mood board*. Une *mood board* ou « planche de tendance », c'est un support où nous allons pouvoir jeter en vrac tous les éléments graphiques qui vont faire de notre application un produit fini. Nuancier de couleur, polices de caractères, icônes, boutons, échantillons d'images, autant d'éléments qui définiront véritablement l'identité graphique de votre app Fig.3.

Voilà une étape sympa et créative où vous allez pouvoir trouver le style qui vous convient, icône flat ou avec effets ? Aplats de couleur, dégradés ou textures ? Etc. Une fois vos choix graphiques posés essayez de bien catégoriser les différents éléments que vous allez créer. D'une part les éléments de navigation actifs (les boutons) et d'autre part les éléments purement informatifs (pictogrammes, flèches, etc.). Une dernière catégorie moins importante est constituée d'éléments décoratifs, qui ne servent qu'à ajouter une plus-value sur le design, mais n'ont pas de rôle fonctionnel dans l'application. Il n'existe que deux manières de dissocier ces éléments au sein de l'app : le contexte et le design.

Une icône peut potentiellement avoir un rôle de bouton ou uniquement d'indication. L'utilisateur doit être capable de comprendre quelle est la nature d'un élément simplement en fonction du contexte.

Exemple : si une icône apparaît sur une barre d'outils contenant uniquement des boutons il y a fort à parier que mon icône soit également un bouton. Pourtant parfois le contexte ne suffit pas à comprendre la nature d'un élément. Dans ce cas il est important que le design de l'élément soit un indice sur sa nature. Cet indice peut être donné par la couleur, la forme du conteneur, la typographie, etc. Exemple : dans Collecto, hormis quelques exceptions, tous les boutons sont contenus dans un cercle ou un carré.

Il existe beaucoup de sites qui proposent des banques d'icônes gratuites (The noun project, iconMonstr, etc). Des outils comme Metro Studio permettent aussi de personnaliser et générer ses icônes à partir d'une banque très fournie. Vous trouverez aussi des outils en ligne pour bien choisir vos couleurs et créer les bonnes associations (Adobe Kuler, Flat UI colors, etc.)

La cible et la thématique de l'application sont des données primordiales à prendre en compte lors de vos choix graphiques. C'est ce qui va définir la direction artistique de votre projet. Si vous hésitez, si vous manquez d'inspiration ou encore si vous ne savez pas quel style appliquer à une thématique liée à votre projet, il est conseillé de faire un peu de veille avant de se lancer. De nombreuses app n'atteignent pas leurs cibles à cause d'une direction artistique hasardeuse. Par exemple une application sportive qui emprunterait sans le vouloir aux codes du luxe (typo manuscrite, couleurs noires et or, etc.) pourrait bien passer à côté de son public. Au contraire si la thématique de l'app transparaît à travers un design juste et équilibré vous aurez de fortes chances de rencontrer votre cible.

Sortir un peu la tête de son projet pour consulter d'autres app ne peut être que bénéfique. Cela vous permettra de voir ce qui se fait chez le concurrent et de vérifier que vous usez des bons codes. Certains sites permettent de booster l'inspiration tels que pttrns.com ou uxarchive.com, vous pourrez y trouver des centaines d'interfaces archivées par catégories. Certains agrégateurs de contenus tels que Dribble, ou Pinterest sont également assez riches en matière d'UI Fig.4.

Enfin, essayez de penser dès maintenant aux différentes transitions et animations qui interviendront dans votre application. Quelles transitions choisirez-vous entre deux pages ? Entre deux rubriques ? Quelles seront les réactions des boutons au survol ? Vous pouvez bien sûr faire le choix de n'utiliser que des animations système. Gardez toutefois à l'esprit que c'est par toutes ces petites choses qui peuvent vous paraître insignifiantes que votre application se démarquera de ses concurrentes. Il est donc conseillé de personnaliser vos animations, et ce, en cohérence avec la thématique de votre projet et donc en harmonie avec vos premiers éléments de chartes.

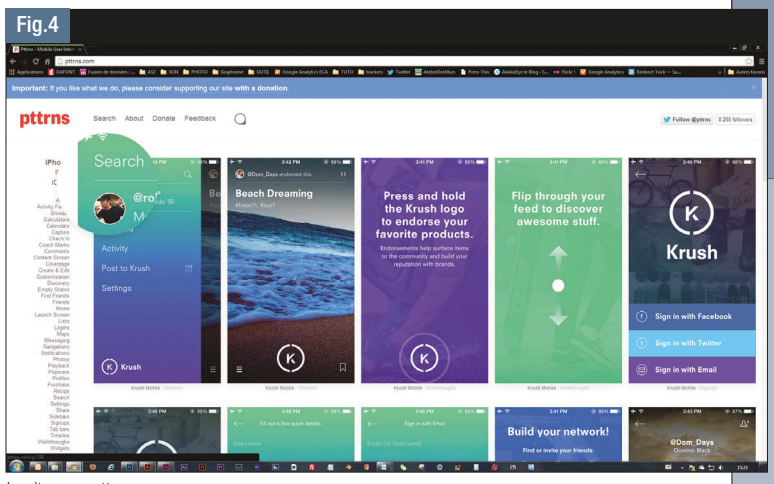
Des sites comme guifff.com ou le tumblr **ui-animations** vous aideront à trouver l'inspiration en matière d'animation.

L'habillage de votre application ne requiert pas obligatoirement une charte graphique à propre-

Fig.3

La mood board de Collecto

Fig.4



Le site www.pttrns.com

ment parler. Toutefois un petit document qui récapitule les principales règles graphiques est loin d'être superflu, surtout si vous travaillez en binôme avec un développeur qui sera chargé de l'intégration de votre créa. Voici les principaux points qu'il vous faudra traiter dans ce doc :

- Typographie. Quelles polices utiliser où et quand ? Quel style (light, regular, etc.) ? Quelle taille ?
- Couleurs. Quel code couleur appliquer dans quelles circonstances ?
- Éléments graphiques. Listing des éléments graphiques, icônes et autres, lesquels utiliser et où ?
- Règle de mise en page, grille, placement des éléments.

Une façon efficace de réunir ces informations sera de commenter quelques pages de maquettes représentatives de l'app en pointant les règles graphiques importantes Fig.5.

Mais pour ce faire encore faut-il avoir des maquettes...

La maquette

La phase de maquettage est une des plus sympathiques pour le designer, car elle représente l'aboutissement de la partie graphique du projet. C'est là que l'on voit se dessiner précisément pour la première fois ce à quoi ressemblera l'application définitive. En revanche pour certains développeurs désignant eux-mêmes leurs apps, cette étape se fera parfois dans la douleur.

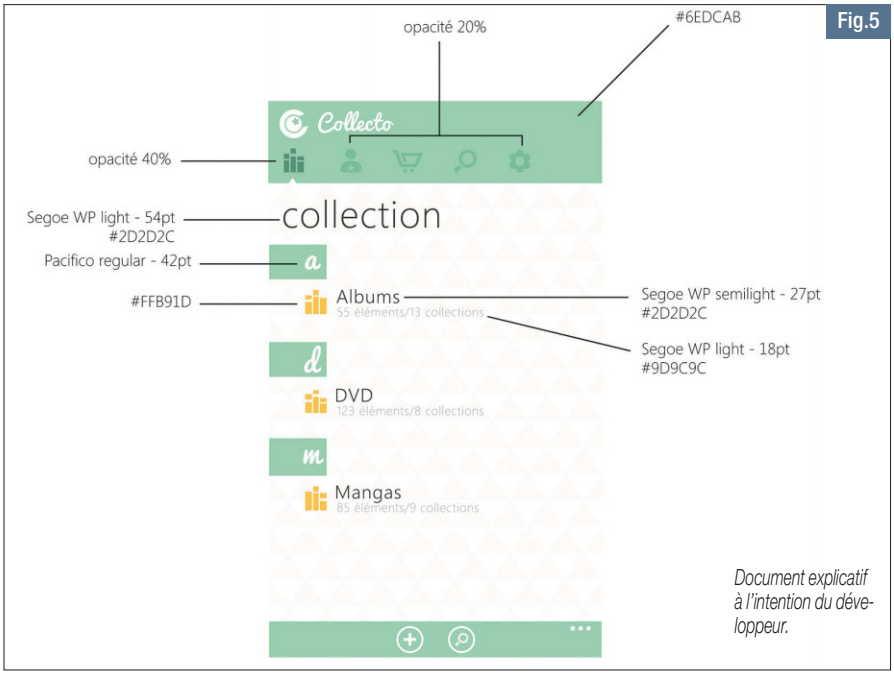
À ces derniers je promets que si les différentes étapes de la création ont été réalisées avec soin, le maquettage leur semblera aussi facile qu'une peinture au numéro. Toute la structure de l'app est déjà là, les choix hiérarchiques sont faits, les zones de contenus placées et définies, les éléments graphiques choisis, créés et validés. Tout ce qu'il vous reste à faire c'est prendre ces éléments et les placer selon le zoning défini Fig.6.

Si vous avez la chance de travailler avec un seul outil pour créer à la fois mockups, moodboard et maquettes, vous aurez l'opportunité intéressante de reprendre vos fichiers et de remplacer simplement les éléments filaires du mockup par ceux de votre moodboard. Pour ma part je travaille toutes ces étapes sous Adobe Illustrator, en vectoriel, ce qui me permet de ne pas me préoccuper des éventuelles problématiques de redimensionnement d'images d'un plan de travail à l'autre.

Concernant les autres outils de création, voici la liste des plus notables :

Adobe Photoshop, Gimp, Photofiltre, Microsoft Expression Design (vectoriel), Paint.Net, Inkscape (vectoriel)

Fig.5



Document explicatif à l'intention du développeur.

Collecto® David CATUHE, Benjamin LAUNAY

Fig.6

Écrans de maquette de l'application Collecto

Collecto® David CATUHE, Benjamin LAUNAY

Le découpage

La phase de découpage n'est pas la plus fun qui soit, mais elle est indispensable à l'intégration. Je ne m'attarderai pas sur cette étape, sachez simplement que si vous travaillez avec Photoshop ou Illustrator, il existe des scripts qui permettent d'automatiser l'export de tous les éléments de vos maquettes en sprites png. Leur usage nécessite de travailler la maquette avec une méthodologie un peu particulière, mais vous gagnerez un temps précieux à l'export. De plus ces scripts vous permettront de nommer automatiquement tous vos sprites à partir des intitulés de calques. Un nommage propre facilitera beaucoup le travail de l'intégrateur. Vous pouvez retrouver dès maintenant un tutoriel détaillé « exporter automatiquement vos sprites png » sur mon blog www.aieaieeye.com, rubrique formation.

Raccourcis vers le tutoriel

Conclusion

Outre le fait d'organiser et de faciliter votre travail ainsi que le workflow avec vos collaborateurs, cette méthode de travail composée d'étapes bien distinctes sera bien perçue par votre client. Cela vous permettra de valider le projet avec lui étape par étape et d'éviter ainsi les retours de correction intempestifs et l'effet cascade désastreux qu'ils pourraient déclencher.

Prenez le temps pour vous poser les bonnes questions à la fin de chaque étape de création :

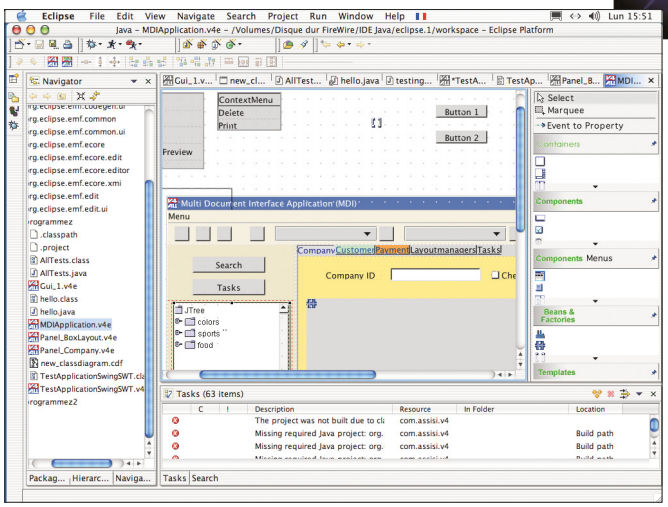
- Mon architecture est-elle objectivement logique ? Peut-on la comprendre sans connaître le projet ?
- La structure de mes pages et mes groupes de contenus reflètent-ils bien l'architecture de mon application ?
- Le style graphique que j'ai choisi est-il cohérent avec la thématique de mon projet ?
- Mes éléments graphiques sont-ils facilement interprétables ?

Posez-vous sans arrêt la question de la cohérence. Enfin n'hésitez pas à mettre votre app en situation en la faisant tester par vos amis, vos collègues, vos grands-parents, etc. Il n'y a que de cette façon que vous pourrez vous assurer que votre application est abordable et compréhensible par un large panel d'utilisateurs.

Benjamin **Launay** — *Designer UI/UX chez Naviso Dev* — www.naviso-dev.com

Timeline : 1998-présent-futur

Objet : Eclipse



Eclipse printemps 2003.

Eclipse est une des plateformes les plus connues dans le monde des développeurs : elle est principalement utilisée pour du développement avec le langage Java, mais aussi pour la construction d'applications clients riches. La partie émergée de l'iceberg est l'outil de développement qu'un grand nombre de développeurs utilisent au quotidien. En réalité Eclipse est un projet et une organisation bien plus vaste. Je vous propose dans cet article de faire un tour d'horizon de l'écosystème d'Eclipse. Nous verrons dans un premier temps un historique de la plateforme. Ensuite, je vous présenterai la fondation Eclipse et le rôle qu'elle a. Nous verrons également les différents projets de cet écosystème. Enfin nous essaierons de voir comment se dessine le futur de l'IDE («Integrated Development Environment»).

HISTORIQUE

La genèse

Au milieu des années 1990 le paysage des outils de développement n'était constitué que de plateformes commerciales. Les principaux éditeurs étaient :

- Symantec avec Visual Café,
- Borland avec Jbuilder,
- IBM avec Visual Age for Java.

A cette période on assistait également à l'émergence des serveurs d'applications conçus pour découpler la partie serveur et la partie cliente d'une application, et faciliter les déploiements et mises à jour. Ces serveurs d'applications étaient :

- Websphere application Serveur d'IBM,
- Weblogic application Server de BEA,
- iPlanet de Sun.

C'est en 1998 qu'IBM se lance dans la création d'une plateforme d'outils de développement autour du langage Java. IBM avait conscience qu'il fallait un écosystème dynamique autour de la plateforme, pour qu'il y ait une adoption massive de l'outil. Pour créer ce contexte dynamique, IBM avait besoin de fonds. Face aux réticences des partenaires, IBM décida en novembre 2001 de mettre le projet sous licence open source en espérant créer la dynamique nécessaire à l'adoption du produit. IBM fonde le consortium Eclipse ainsi que le site Eclipse.org. Les premiers membres sont :

- Rational Software,
- Together Soft,
- WebGain,
- Borland.

Pour faire partie du consortium, il faut respecter certaines règles (en théorie). Assurer la promotion en interne et en externe du produit, et bâtir des produits sur celui-ci. L'organisation générale autour de l'outil se fait de la manière suivante :

- Le code source est géré par la communauté open source,
- Les aspects marketing et commerciaux sont pris en charge par le consortium.

C'est une nouvelle approche (innovante) du modèle open source. C'est-à-dire que d'une part il est basé sur une plateforme ouverte et gratuite, d'autre part des sociétés commerciale qui y sont impliquées encouragent son utilisation et la création de produits à dessein commercial. En 2003, la première version majeure d'Eclipse voit le jour. Malgré une adoption massive par les développeurs, Eclipse reste perçue comme une plateforme, certes open source, mais restant sous le contrôle d'IBM. Cette perception induit un doute chez les principaux acteurs du marché, et ne les incite pas à bâtir une stratégie autour d'Eclipse. IBM devait éliminer cette perception en rendant Eclipse indépendante. C'est pour cette raison qu'IBM créa la fondation Eclipse. L'annonce de la création fut faite en 2004 lors de la conférence dédiée à la plateforme : EclipseCon 2004.

La fondation Eclipse fut un succès. La première version livrée sous l'égide de la fondation fut la version 3.0 suivie rapidement après de la version 3.1. L'engouement pour cette version fut total. Le nombre d'adhésions des membres a été exponentiel. En décembre 2004, IBM Rational a remanié l'ensemble de ses produits pour les porter sur la plateforme Eclipse. Lors du EclipseCon suivant (2005) un nombre important d'annonces a été fait :

- Lancement de Eclipse RCP (Rich Client Platform)
- Web tools plateforme
- Business Intelligence Reporting Tools

Cette liste n'est pas exhaustive.

Les versions

Jusqu'à janvier 2006 les versions n'avaient pas de nom particulier à part leur numéro de version. C'est lors de la version 3.2 qu'un nom de code fut donné à chaque nouvelle release de la plateforme :

- Eclipse 3.2 : Calipso
- Eclipse 3.3 : Europa
- Eclipse 3.4 : Ganymède
- Eclipse 3.5 : Galileo
- Eclipse 3.6 : Helios
- Eclipse 3.7 : Indigo
- Eclipse 4.0 : Juno
- Eclipse 4.3 : Kepler
- Eclipse 4.4 : Luna

Depuis 2007, le plan de développement prévoit une version majeure au mois de juin et deux versions intermédiaires en septembre et février.

La version courante est nommée Kepler; elle est sortie le 27 juin 2013. La prochaine version nommée Luna est prévue pour le mois de juin 2014.

LE PROJET ECLIPSE ET LA FONDATION ECLIPSE

Il est important de bien comprendre la distinction entre le projet Eclipse et la fondation du même nom.

Le projet Eclipse

Le projet Eclipse rassemble des personnes indépendantes ou faisant partie d'une organisation. L'objectif étant d'établir une collaboration

autour de la construction d'une plateforme open source. Cette plateforme peut se voir selon 2 points de vue différents :

L'IDE Eclipse

L'IDE eclipse est un environnement de travail pour les développeurs. Il existe plusieurs versions packagées de l'outil. Chaque version propose un ensemble d'outils pré-intégrés (plugins déjà installés). On peut trouver comme package :

- Eclipse JEE
- Eclipse STS (Spring Tools suite)
- Eclipse Java
- Obeo Designer
- Eclipse C/C++

Cette liste n'est pas exhaustive

Eclipse RCP

Eclipse RCP (Rich Client Platform) n'est plus une plateforme de développement mais une application de type progiciel à destination d'utilisateurs finaux. On peut retrouver cette plateforme sur les projets suivants :

- Business Intelligence and Reporting Tools (BIRT)
- Lotus Notes
- Eclipse Platform
- Plugin Development Editor
- Data Tools Platform

Parmi ces projets, on retrouve une liste impressionnante de plugins. On peut retrouver la liste exhaustive en allant sur le marketplace de l'outil (http://goo.gl/hBJxtb)

La fondation Eclipse

La fondation Eclipse est une organisation à but non lucratif. Elle est financée par ses membres. Les membres de l'organisation sont distingués selon une classification précise comprenant 5 niveaux:

- Associé
- Solution
- Entreprise
- Strategic
- Commiter

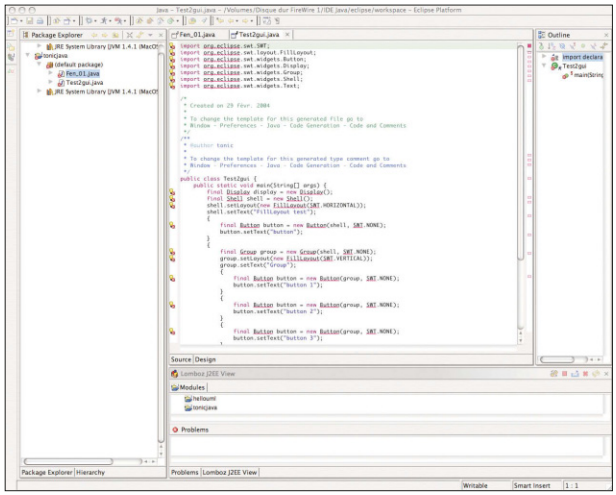
Vous trouverez le détail de chaque niveau en suivant ce lien : http://goo.gl/hzksDU

La fondation est gouvernée par un "board Directors" constitué de 16 personnes au profil hétérogène. Vous trouverez la liste exhaustive des membres en suivant ce lien: http://goo.gl/MG42iN

Cette fondation emploie à plein temps une équipe dédiée à l'organisation et la gestion de la communauté Eclipse.

Une chose à retenir est que parmi ces personnes on ne retrouve pas de commiter sur le projet. Ce n'est pas l'objectif de ce staff.

Les 4 principaux services rendus par cette équipe sont :



Eclipse début 2004

La gestion de l'infrastructure du projet

Actuellement l'infrastructure se compose des éléments suivants :

- Gestion d'un repository Git,
- Une base Bugzilla,
- Gestion de mailling lists et de forums dédiés au projet,
- Gestion du site de téléchargement,
- Gestion du site web.

La gestion de la propriété intellectuelle

La fondation Eclipse encourage l'utilisation d'Eclipse par des sociétés éditrices de logiciels comme base de leurs produits. Etant donné que le projet est diffusé la licence EPL (Eclipse Project License), ce cadre est possible.

Donc le rôle de la fondation, dans ce cadre, est de s'assurer que l'ensemble des développements réalisés utilisent le bon *copyright* ainsi que la bonne licence (EPL).

Elle se charge également de faire signer à tous les contributeurs une charte (http://goo.gl/ExpEdq) stipulant que toutes leurs contributions sont issues de travaux originaux et qu'ils sont bien diffusés sous la License EPL. Si le développeur travaille pour une organisation, la fondation se charge de faire signer la même charte à l'organisation. Dans le cadre de la gestion de la propriété intellectuelle, l'équipe se charge de vérifier le code mis en oeuvre en dehors du processus de développement d'Eclipse.

Le code est alors analysé afin de vérifier sa provenance et s'il est compatible avec la licence d'Eclipse (EPL).

L'objectif étant de pouvoir garantir aux projets commerciaux basés sur Eclipse que leurs outils peuvent être distribués en toute sérénité.

Le support à la communauté de développement

L'équipe de la fondation met en place un processus standard de développement autour d'Eclipse. Ce processus vise à aider les nouveaux projets à démarrer au sein de la communauté. Il vise également à assurer que tous les projets évoluent de manière équitable dans l'écosystème du projet Eclipse, de façon transparente, ouverte et basée sur la méritocratie. Annuellement, la fondation organise la coordination de la future release avec deux objectifs :

- s'assurer que tous les projets de la release seront disponibles au même moment.
- d'effectuer une phase de test globale avant la version finale afin d'assurer la qualité des livrables dans son ensemble.

Le développement de l'écosystème du projet Eclipse

Afin d'assurer la promotion et le développement de l'écosystème d'Eclipse, la fondation organise différents évènements :

- **EclipseCon** : grande conférence annuelle autour du projet Eclipse et de son écosystème.
- La mise en ligne de ressources : **Marketplace** ou **la chaîne youtube** du projet.
- Une réunion bisannuelle rassemblant les membres de la communauté.

LE FUTUR D'ECLIPSE

Présenter le futur d'Eclipse n'est pas forcément évident, en tous cas de manière objective.

Les éléments sur lesquels on peut se baser pour donner une réponse sont nombreux.

On peut tout simplement regarder les chiffres présentés dans le bilan annuel de la fondation, ou alors s'adresser à la communauté de développeurs pour connaitre leurs habitudes de travail. Donner mon avis personnel ne serait pas du tout représentatif et intéressant.

Je vous propose donc de vous présenter plusieurs points de vue :

- Les chiffres du bilan annuel,
- Le résultat d'un sondage que j'ai réalisé.

A partir de ces éléments chacun pourra se faire une idée sur la question.

Les chiffres de la fondation

Le nombre de membres dans la fondation

Chaque année la fondation présente un bilan selon différents critères. Le critère qui me semble pertinent, par rapport à la question que l'on se pose est le nombre de membres (de tous horizons) dans la fondation.

Ci-dessous les chiffres issus du bilan :

2004	2005	2006	2007	2008	2009	2010	2011	2012	Mars 2013
50	91	130	157	180	172	165	171	186	190

On constate que depuis 2004, le nombre est en constante progression jusqu'à 2008. Après cette date, on peut voir une baisse jusqu'en 2012 où le

nombre de membres dépasse le dernier pic de 2008. Si l'on prend une moyenne du nombre de membres sur les dernières années, on voit que l'adhésion est finalement plutôt stable.

Projets actifs

Ci-dessous l'évolution du nombre de projets actifs au fil des années :

2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
13	18	24	41	68	89	113	141	152	181	197	177

Le nombre de projets actifs est en constante progression jusqu'en 2011, l'année d'après il y a eu une baisse significative.

Commiters actifs

Ci-dessous l'évolution du nombre de commiters actifs au fil des années :

2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
77	89	123	187	319	413	520	577	580	641	650	613

A l'instar du nombre de projets actifs, la progression est quasi constante jusqu'en 2011 puis une baisse significative est constatée en 2012.

Projets par release

Ci-dessous l'évolution du nombre de projets par release de la plateforme Eclipse

Eclipse 3.0	3	Eclipse 3.1	7
Callisto	10	Europa	21
Ganymede	23	Galileo	33
Helios	39	Indigo	62
Juno	72		

Les chiffres donnés ci-dessus montrent que l'IDE s'est beaucoup enrichi au cours des années et que la communauté a été très productive sur la plateforme.

Sondage

Afin d'avoir une vision globale de la communauté sur Eclipse, j'ai lancé un sondage autour de moi en posant les questions suivantes :

- Quel est votre IDE quotidien (pro/perso) (si différent, pourquoi?)
- Quel IDE utilisiez-vous en 2010, 2011, 2012, 2013, 2014, et dans le futur.
- Si vous deviez donner 3 avantages concernant Eclipse par rapport à la concurrence directe (Idea, NetBeans)
- Si vous deviez donner 3 inconvénients concernant Eclipse par rapport à la concurrence.
- Quel est le futur d'Eclipse selon vous?

L'objectif était d'essayer de capter une tendance. Le nombre de réponses obtenues ne permet pas d'affirmer la tendance, en revanche le profil des personnes interrogées apporte du crédit aux réponses.

D'une manière générale, ce sont des développeurs professionnels qui utilisent cet outil (ou un autre) dans leur travail de tous les jours ou de manière personnelle.

Même si ce sondage ne suffit pas pour avoir un point de vue objectif, il donne tout de même un état d'esprit, et par conséquent une tendance sur le futur de l'IDE. 16 personnes ont répondu au sondage.

Voilà donc une synthèse des résultats :

- *Quel est votre IDE quotidien (pro/perso) (si différent, pourquoi?)*

La majorité des réponses ont porté sur un autre IDE qu'Eclipse.

- *Quel IDE utilisiez-vous en 2010, 2011, 2012, 2013, 2014, et dans le futur).*

Dans la plupart des cas il y a eu un abandon d'Eclipse entre 2012 et 2013 au profit d'un IDE alternatif.

- *Si vous deviez donner 3 avantages concernant Eclipse par rapport à la concurrence directe (Idea, NetBeans)*

Les principaux avantages mis en avant pour Eclipse sont :

- La gratuité de la plateforme,
- La grande bibliothèque de plugins disponibles,
- La taille de la communauté autour de l'IDE,
- L'aspect relativement standard de l'outil dans l'écosystème des développeurs.

- *Si vous deviez donner 3 inconvénients concernant Eclipse par rapport à la concurrence.*

Les principaux inconvénients rapportés par le sondage sont :

- La lourdeur et la lenteur de la plateforme,
- Paradoxalement, trop de plugins,
- Problème d'intégration avec Maven,
- Pour des développeurs en frontal : instabilité de l'IDE.

- *Quel est le futur d'Eclipse selon vous?*

J'ai eu peu de réponse à cette question. Néanmoins j'ai quelques réponses intéressantes que je vous retranscris ici :

- *"Eclipse est une plateforme (plus qu'un IDE) qui permet via RCP/SWT de développer des applications Desktops"*
- *"Il va continuer à cohabiter avec les IDEs payants pendant un bon moment."*
- *"... On voit que même Google ne sait pas trop sur quel pied danser. D'un côté son éditeur Dart, ou son super plugin AppEngine, basé sur Eclipse. D'un autre côté sa suite Android basée sur IntelliJ..."*
- *"... Créer un environnement de travail spécialisé est la force d'Eclipse..."*

On peut constater au travers de ce petit sondage que l'opinion sur l'IDE Eclipse n'est pas toute blanche ou toute noire. Sur l'ensemble des personnes interrogées, une majorité a abandonné la plateforme.

Les points critiques à l'encontre de la plateforme sont assez récurrents au travers des critiques que l'on peut entendre sur les projets. Par contre sa communauté, sa gratuité ainsi que l'ensemble de ses plugins en font un IDE très populaire.

Il reste difficile de se faire une idée précise de son avenir. On peut le constater ici.

Evidemment, les informations et arguments donnés ici ne sont qu'à titre indicatif et ne constituent pas une vérité absolue. Ce ne sont que des éléments de réflexion.

Les dernières news

J'ai découvert il y a peu de temps une initiative qui permettra, peut-être, de donner une perspective complémentaire à l'outil. Ce projet hébergé pour l'instant par le site kickstarter (https://www.kickstarter.com/projects/283096083/easyeclipse-for-java) propose de bâtir une version sous licence payante de la plateforme Eclipse en concentrant les efforts sur les problèmes récurrents de l'outil (lenteur, qualité des plugins natifs, etc.). On retrouve le modèle sur lequel se base le principal concurrent IntelliJ. Un projet à suivre...

Nous arrivons au terme de cet article. Nous avons pu voir la genèse de la plateforme ainsi que l'organisation de son écosystème. Enfin dans la dernière partie, j'ai essayé de donner des pistes concernant le futur de la plateforme.

Fabrice **Sznajderman**
Senior Java / Scala / Web developer at SFEIR

Abonnement : Programmez, 17, Route des Boulangers, 78926 Yvelines Cedex 9 - 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 : G. Renard, Julien, Simon, Arnaud, L. Rebours, C. Pichaud, S. Lucas, S. Rios, R. Calascibetta, P. Guillerm, M. M. Bertocchi, P-H Gache, C. Peugnet, V. Saluzzo, S. Civetta, N. Thenoz, M. Moizard, R. Masson, A. A. Ettarrouzi, M. Tasliman-ka Sylla, B. Launay, F. Sznajderman, CommitStrip

Crédits couverture : 09-18-12 © DrAfter123 / istock
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.

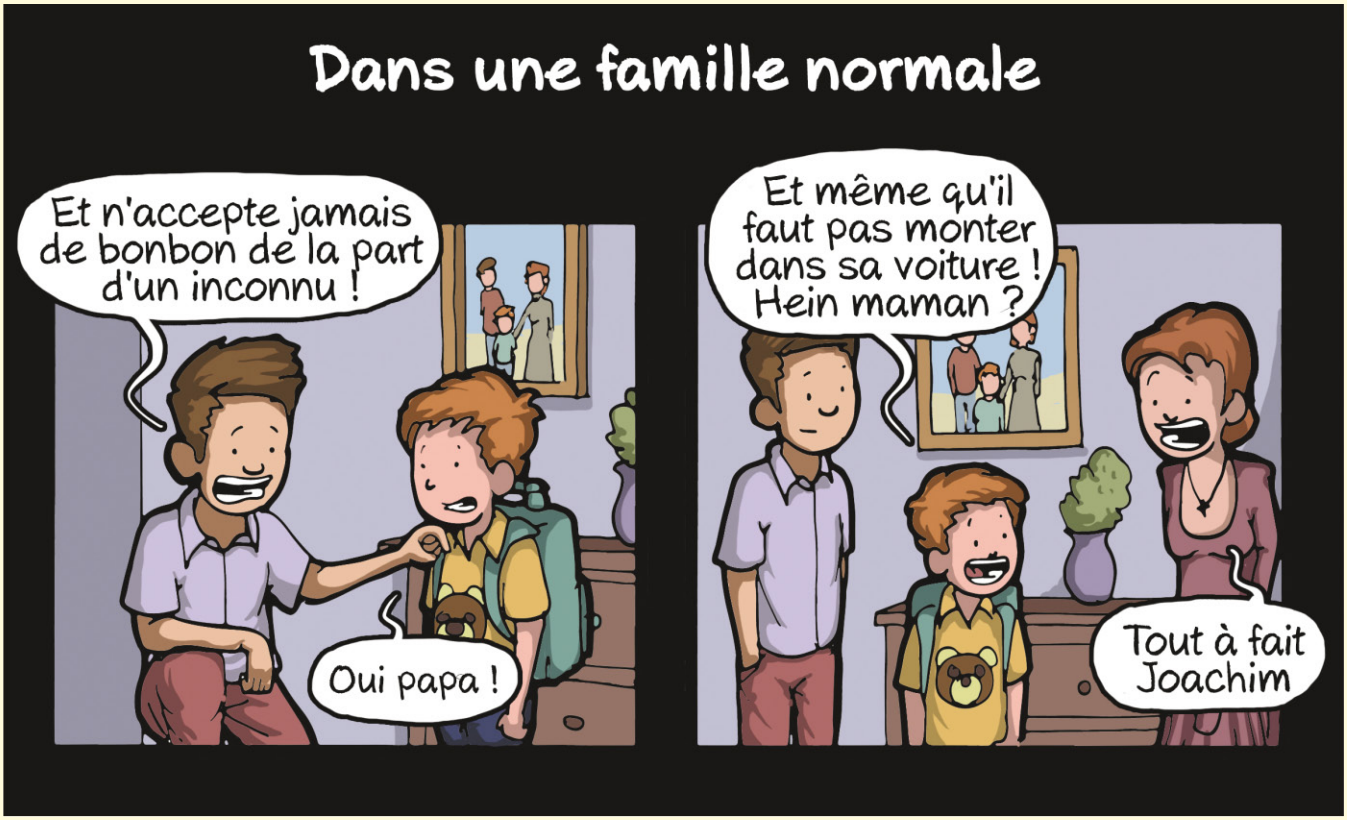
Marketing et promotion des ventes :
Agence BOCONSEIL - Analyse Media Etude

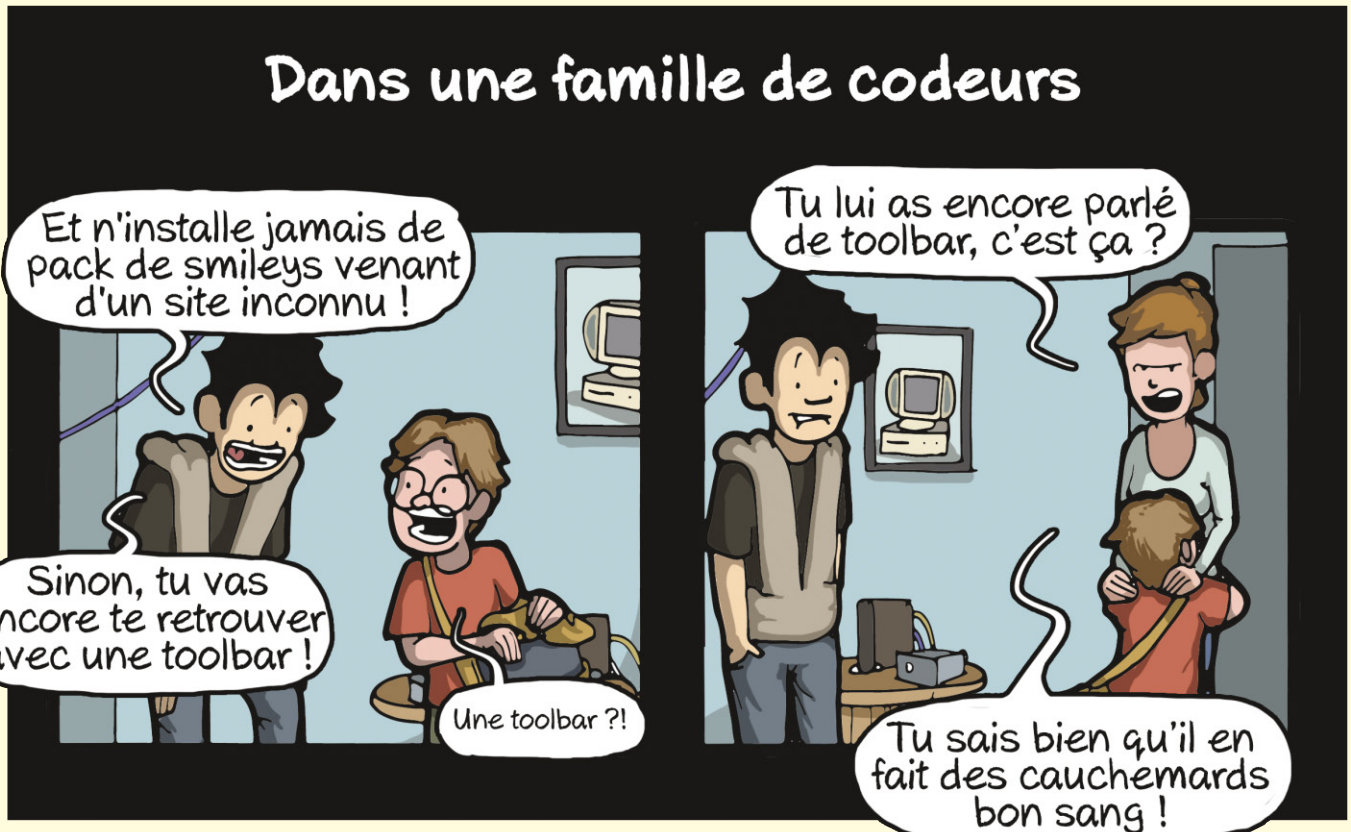
Directeur : Otto BORSCHA oborscha@boconseilame.fr

Responsable titre : Terry MATTARD
Téléphone : 0967320934

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





CommitStrip.com

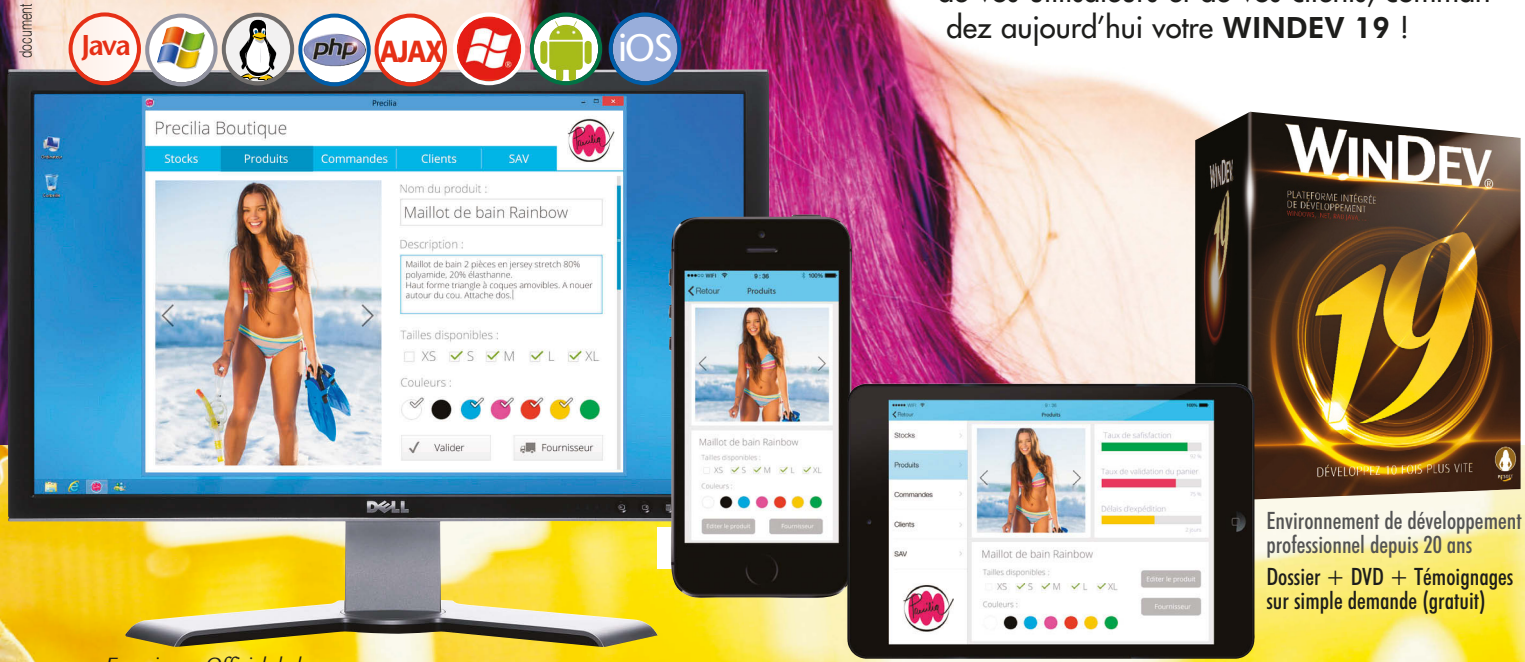
VOUS AUSSI DÉVELOPPEZ 10 FOIS PLUS VITE

Vous méritez le meilleur, vos équipes méritent le meilleur pour être le plus efficace possible, quel que soit le matériel sur lequel vos applications vont fonctionner : **PC**, **tablette**, **smartphone,** en application native ou en **site Internet**.

Incroyable : grâce à **WINDEV 19**, **WEBDEV 19** et **WINDEV Mobile 19**, **les applications que vous écrivez sont nativement portables**.

Vous réutilisez votre code, vos fenêtres, vos états... dans tous les environnements: **Windows**, Mac, Linux, **Android**, iOS (**iPhone**, **iPad**), **Windows Phone**, pour des applications natives et pour des sites, avec les données en local, sur serveur local ou distant, ou encore dans le **cloud**.

Dans l'intérêt de votre entreprise, dans l'intérêt de vos utilisateurs et de vos clients, commandez aujourd'hui votre **WINDEV 19** !



Environnement de développement professionnel depuis 20 ans

Dossier + DVD + Témoignages sur simple demande (gratuit)

Fournisseur Officiel de la Préparation Olympique

WINDEV AGL N°1 en FRANCE

www.pcsoft.fr

Des centaines de références sur le site