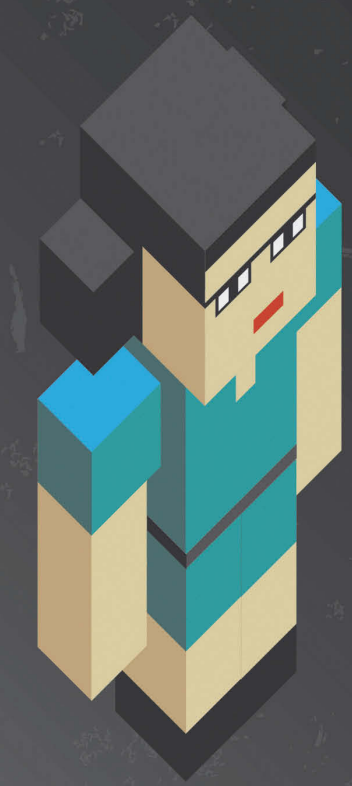
programmez!
#193 - Février 2016
le magazine du développeur

C# 6.0
C# puissance 10

Choisir sa base de données
SQL ou NoSQL

.Net native
Aussi rapide que C++ ?



© AcidLabs

Sécurité, Hacking, Failles, tests...
Les défis du développeur

Les alternatives à Delphi

M 04319 - 193 - 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

ikoula
HÉBERGEUR CLOUD

TROPHÉES DUCLOUD by EuroCloud
MEILLEUR SERVICE CLOUD D'INFRASTRUCTURE 2015

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

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

ENTERPRISE SERVICES

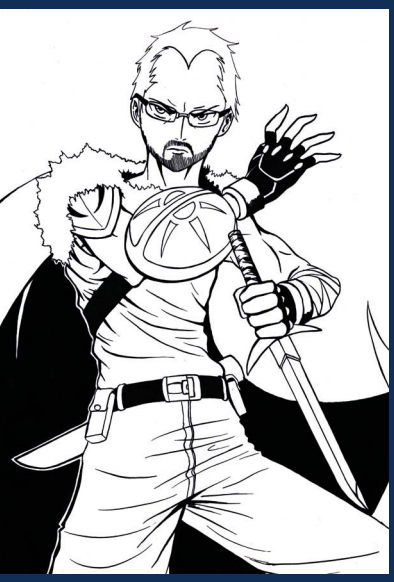
Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

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

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

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

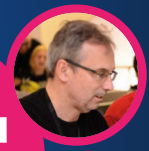


Nouvelle année = bonnes résolutions.

- Comprendre enfin la différence entre C# et Java
- Se mettre à Python
- Faire du DevOps et être agile
- Lire un dictionnaire de design pour comprendre ce que dit le designer
- Ne pas taper le sysadmin quand tout plante
- Ne pas faire une mise en prod le vendredi 17h
- Lire les specs du projet (= tu sais le truc qui ressemble à un livre avec plein de mots dedans)
- Dire « C'est pas faux » quand tu discutes avec des non-geeks
- Ne pas porter le même t-shirt 5 jours de suite (= y'a une invention géniale qui s'appelle une machine à laver)
- Mettre des commentaires dans son code
- Ah il existe autre chose que le NoSQL ?
- Faire du sport (= trouver une école de Jedi)
- Ne pas spoiler les séries
- Les 10 geeks commandements tu respecteras : « dans le doute reboote, si ça rate formate » (finalement, y'en a que 2, merci Les Geeks)
- Devenir éleveur de Pokémon (la preuve, j'ai commencé ce matin)
- Lire Programmez ! chaque mois (= c'est bon pour le cerveau)

François Tonic /
dresseur de Pokémon
ftonic@programmez.com

sommaire



À lire dans le prochain numéro n°194 en kiosque le 27 février 2016

Mon code est-il légal ?
Trouver l'origine des codes sources réutilisés, connaître les licences des codes, les bons réflexes.

Le code est un art
Faire un code propre et bien documenté : oui c'est possible !

Maker
Construire une borne d'arcade avec une Raspberry Pi.

Remix OS

permet d'installer Android sur un PC

Un desktop web écrit en JavaScript ? Oui c'est possible avec

OS.js…

http://os.js.org

Hourra, on peut commander son

Oculus Rift pour 699 €…

par contre, PC surpuissant vendu séparément.

Le

marché PC

n'est pas en grande forme (selon IDC)

Brillo,

le système Google pour les objets connectés est toujours en développement et sur invitation : https://developers.google.com/brillo

La course à l'

Intelligence artificielle

est lancée. Et le vainqueur est...

Ian Murdock

est mort en décembre dernier. Il fut l'initiateur de Debian.

Tu veux d'un système d'exploitation souverain ?...

« Le Gouvernement remet au Parlement, dans les trois mois suivant la promulgation de la présente loi, un rapport sur la possibilité de créer un Commissariat à la souveraineté numérique rattaché aux services du Premier ministre dont les missions concourront à l'exercice, dans le cyberespace, de la souveraineté nationale et des droits et libertés individuels et collectifs que la République protège. Ce rapport précise les conditions de mise en place, sous l'égide de ce Commissariat, d'un système d'exploitation souverain et de protocoles de chiffrement des données, ainsi que les moyens et l'organisation nécessaires au fonctionnement de cet établissement public. » Tel est le premier article de l'amendement N°CL129 déposé à l'Assemblée Nationale par Mme Batho et M. Grandguillaume. L'idée n'est pas réellement nouvelle et après l'échec du cloud souverain français, la proposition étonne...

... ou d'une backdoor ?

Un autre amendement (n°CL92) propose d'imposer une backdoor aux constructeurs de matériel informatique pour permettre l'accès aux matériels (et donc aux données des utilisateurs) dans le cadre d'une enquête, après autorisation d'un juge...

Google veut un Chrome plus simple !

En juillet dernier, les équipes Chromium ont lancé le projet Eraser. Le but est simple : simplifier Chrome. Il s'agira de retirer des fonctions, de faire le ménage dans le code, rendre plus ergonomique l'interface. Par exemple, des fonctions rarement utilisées pourront être supprimées et gagner ainsi quelques pixels et simplifier l'interface générale, donner plus de lisibilité aux paramètres... Mozilla avait annoncé la même chose.

Un vaste chantier :

https://code.google.com/p/chromium/issues/detail?id=512852

Et Java 9, il en est où ?

Il devait sortir en 2016. Le projet OpenJDK est prudent et le décale à mars 2017...

date	étape
fin mai	fonctionnalité complète
août 2016	on lance les tests complets
octobre	on fixe les bugs
début décembre	on scrute attentivement le code
fin janvier 2017	dernière version finale avec la sortie
23 mars 2017	sortie officielle

L'INDEX TIOBE DU MOIS

01/2016	01/2015	Tendance	Langage	%	Évolution
1	2	^	Java	21.465 %	+5.94 %
2	1	v	C	16.036 %	-0.67 %
3	4	^	C++	6.914 %	+0.21 %
4	5	^	C#	4.707 %	-0.34 %
5	8	^	Python	3.854 %	+1.24 %
6	6		PHP	2.706 %	-1.08 %
7	16	^^	VB.NET	2.582 %	+1.51 %
8	7	v	JavaScript	2.565 %	-0.71 %
9	14	^^	Assembleur	2.095 %	+0.92 %
10	15	^^	Ruby	2.047 %	+0.92 %

Java redevient le plus populaire dans les recherches devant C. En un an, Objective-C c'est totalement effondré il est désormais 18e au classement. Swift se classe devant son « ancêtre » et se positionne à la 14e place. Pour le reste, peu de surprises sauf l'assembleur qui est 9e...

Rappelons que l'index se base sur les recherches sur les moteurs et ne représente pas l'usage réel du langage par les développeurs.

Dans le futur, vous serez (toujours) développeur

Notre sondage « votre futur métier... » a montré que vous restez majoritairement développeur, même demain (55 %). Pourtant, nous avions des profils sympas pour le futur, certains vous ont plu :

- Charmeur d'objets et d'ordinateurs (7 %)
- Maître du boulier chinois (7 %)
- Aiguilleur de drones 6 %
- Éleveur d'animaux-robots 6 %

Les autres carrières ne vous intéressent pas réellement (réparateurs, boutique pour maker / geek, pilote de drones-livreurs, conseil en VR, installeur d'objets connectés).

WEBDEV®

NOUVELLE VERSION 21

CRÉEZ FACILEMENT DES SITES «RESPONSIVE WEB DESIGN»

ACCÉDANT À VOS BASES DE DONNÉES

Fournisseur Officiel de la Préparation Olympique

Rendez vos sites «Mobile Friendly».

WEBDEV 21 vous permet de rendre facilement vos sites «Mobile Friendly».
Les sites que vous créez sont ainsi mieux référencés par Google.
Responsive Web Design et Dynamic Serving sont à votre service dans WEBDEV 21.

WEBDEV** est compatible avec **WINDEV

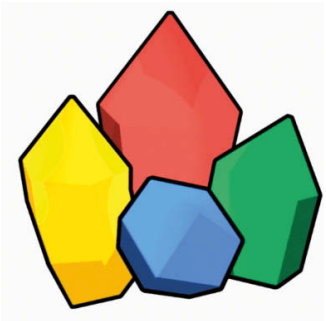
DÉVELOPPEZ 10 FOIS PLUS VITE

www.pcsoft.fr

Des centaines de témoignages sur le site

février

DevFest Paris, le 5 février !



Très gros succès à Nantes, la DevFest arrive à Paris. Cette grande journée de développement et de technologies est organisée par le Google Developer Group de Paris. La journée se déroulera de 8h à 20h. 700 personnes attendues et plus de 25 conférences !
Lieu : La Grande Crypte, Paris.
Site officiel : http://devfest.gdgparis.com

mars

NIDays 2016 : 10 mars

National Instruments organise sa journée annuelle à Paris le 10 mars prochain. Occasion pour faire le point sur les différents outils, l'instrumentation, l'informatique industrielle et embarquée. La journée sera rythmée par de nombreuses conférences.
Grand rendez-vous de la journée, deux coupes seront organisées, la Coupe NXT et la Coupe RIO, dont le thème commun est la robotique.
Pour plus de détails : http://france.ni.com/nidays/coupes-robotiques
Site officiel : http://france.ni.com/nidays

avril

Devoxx 2016 : du 20 au 22 avril 2016

Réservez déjà les dates pour venir au plus grand événement Java en France. Cette année encore, la conférence promet beaucoup, avec de nombreux thèmes abordés : architecture, sécurité, cloud, core java, les langages de la JVM, méthodologie, mobilité, html 5...
Site : http://www.devoxx.fr/

Quelques dates à retenir

Meetup Swift – RXSwift & TBD : le 15 mars, meetup à Paris autour du langage Swift et de RXSwift.
Site : http://www.meetup.com/fr-FR/swiftparis/events/227455210/

Evolve16 : la conférence développeur de Xamarin se déroulera en Floride du 24 au 28 avril. De nombreux thèmes seront abordés : design, compilation, intégration, sécurité, tests, monitoring...
https://evolve.xamarin.com
La conférence Ncrafts 2016 aura lieu cette année les 12 et 13 mai

MUG Strasbourg : les prochaines réunions

Le Microsoft User Group de Strasbourg organise régulièrement pour la communauté des conférences autour des technologies Microsoft. Les prochains événements prévus sont :

- Février : conférence sur le langage F#
- Mars : conférence sur TypeScript et Angular 2

A l'heure où nous publions, les dates précises ne sont pas encore connues.

- Facebook : https://www.facebook.com/groups/MugStrasbourg/?fref=ts
- Twitter : https://twitter.com/MUGStrasbourg

PHP Tour 2016 à Clermont-Ferrand

L'AFUP pose les Elephpants à Clermont-Ferrand en mai prochain. Une occasion de réunir la communauté, l'écosystème et les acteurs du monde PHP. L'appel aux conférences sera cloturé le 20 février. Site : http://event.afup.org
Microsoft TechDays 2016 : les fameux TechDays se dérouleront cette année en octobre et sur deux jours

Quelques meetups

- 3 février - Bordeaux : PHP Meetup sur « Haxe pour les développeurs web ». site : http://aquinum.fr/l-agenda/les-evenements-aquinum/evenement/467-php-meetup-10-haxe-pour-les-developpeurs-web
- 6 février – Paris : NodeSchool Paris de 10h à 18h30 pour apprendre les bases de JavaScript. Site : http://www.meetup.com/fr-FR/NodeSchool-Paris/events/227823497/
- Meetup Drupal en France : plusieurs meetup autour de Drupal auront lieu en février. Pour découvrir les dates et les villes concernés : http://drupalfr.org/prochains-evenements

MAKERFAIRE PARIS 2016 : LES 30 AVRIL & 1ER MAI

Cette année encore, MakerFaire se tiendra durant la Foire de Paris, à la porte de Versailles. Tout Maker peut proposer son projet et être présent durant l'événement. En 2015, ce sont plus de 35 000 visiteurs qui ont parcouru les allées. Site : http://www.makerfaireparis.com
Cette année, deux mini Maker Faire seront organisées en France :

- Saint-Malo : 9 & 10 avril
- Rouan : 3 & 4 juin

FAÇONNONS ENSEMBLE L'INTERNET DES OBJETS

NIDays

Rejoignez plus de 1 200 innovateurs issus de secteurs industriels variés, venez échanger avec les équipes NI et découvrez comment les avancées technologiques convergent pour créer un monde plus intelligent et plus connecté, reposant sur des systèmes conçus par logiciel.

Paris, Palais des Congrès
Jeudi 10 mars 2016

Inscription gratuite sur **ni.com/nidays.**
Suivez-nous sur Twitter : **@NIFrance** et en live avec **#NIDays.**

NATIONAL INSTRUMENTS™

Le profil du développeur 2016

Suite à notre grande enquête, Programmez ! vous propose un profil type du développeur sur les technologies, les plateformes utilisées, les langages.

La rédaction

Source de l'enquête : sondage mené en 2015. Les résultats se basent sur 110 répondants.

Résumé

Vous êtes majoritairement un homme (6,6 % de femmes) avec un âge moyen de 30,3 ans. Vous habitez plutôt en province (50,9 %) contre 36,36 % en Ile de France et 12,74 % à l'étranger. Votre formation est, sans surprise, un bac+5 (ou +) pour 44,45 % d'entre vous, 20 % en bac+2. Mais vous êtes tout de même quelques autodidactes (11,8 %). Elément intéressant, vous êtes 15 % à être venus à la programmation avec une formation non informatique.

Pour 68 % des répondants, le développement est votre activité principale (et pour 15,45 % une passion). Vous vous définissez comme développeur sans autre forme. Cependant, vous êtes 12,72 % à vous voir comme un développeur full-stack et 11,81 % comme développeur Web.

Vous travaillez pour moitié en entreprise. Vous êtes 14,45 % à être indépendant et 15,45 % en SSII. Et vous ne vous voyez pas changer de profil dans les 5 ans (42 %) mais le métier d'architecte logiciel ne vous laisse pas insensible (pour 16,36 %).

Sans réelle surprise, vous êtes 72 % à travailler sur un poste Windows et à utiliser (minimum 50 %) un IDE, des frameworks, des bases de données, un référentiel de code. Et les applications développées sont des applications côtés serveurs, desktop, mobile et applications Web. Les langages phares que vous utilisez sont, là encore sans aucune réelle surprise (dans l'ordre) : HTML / CSS, SQL, JavaScript, PHP, C#, Java et C / C++. A noter la bonne tenue de Python (16 %) et même de Delphi, 11 %...

Si vous développez sur mobile, vous développez pour 45 % pour Android, mais, surprise, Windows Phone arrive 2e avec 30 % des répondants, iOS est juste derrière... Et le développement natif pèse 41 % des développements mobiles. Mais si vous devez utiliser un outil multiplateforme, vous êtes plutôt Cordova que Xamarin... D'autre part le Cloud Computing n'est pas étranger à votre quotidien, vous êtes 57 % à utiliser un ou plusieurs services (pas forcément pour le développement). Mais aucun fournisseur Cloud ne détache réellement dans vos choix. Mais pour demain, vous êtes très attentifs et intéressés par le Cloud Computing et les IoT et maker, deux mondes technologiques arrivant en tête des technologies que vous allez utiliser ou regarder dans les prochains mois.

LES CHIFFRES CLÉS À RETENIR

6,6 % de femmes
30,3 ans
50,9 en province
46 % avec un profil pur dév.
72 % sous Windows
HTML / SQL / JavaScript le trio infernal
21 % utilisent du NoSQL
Swift intrigue à 15 %
Delphi n'est pas mort pour 11 %
57 % de projets Web
Beaucoup de futurs makers

DANS LE DÉTAIL

Votre profil de développeur...

actuellement	dans 5 ans
Développeur 46,36 %	Rester développeur 42,72 %
Chef de projet 13,63 %	Devenir architecte logiciel 16,36 %
Dév. Fullstack 12,72 %	Devenir chef de projet 10 %
Dév. Web 11,81 %	Devenir développeur mobile 10 %
Autre 15,48 %	Autre 20,92 %

Votre poste de travail

Vous travaillez sur Windows (72 %), puis suivent Linux (12,72) et OS X (10 %). Ce n'est pas une réelle surprise. Linux ne se défend pas si mal...

Vous utilisez de nombreux outils dont :

QUELLES TENDANCES DE SALAIRES POUR 2016 ?

Le cabinet Hays a publié de nouvelles données sur les salaires dans le monde informatique. On constate que le développeur évolue finalement assez peu en salaire même si Hays donne une fourchette de 38 /43 k€ pour 0 à 3 d'expérience et en île de France. Le niveau est plutôt bac+5. Une décote importante est à prévoir pour la province et selon le niveau de formation. Cependant, des profils rares ou très experts pourront négocier des salaires moyens supérieurs.

	0 – 3 ans	3 – 5 ans	5 – 8 ans	+8 ans
Ingenieur test QA	28 /32	32 /35	35 /45	45 /55
Ingenieur etudes & developpement	38 /43	43 /50	50 /60	60 /65
Lead technique	NS	NS	50 /60	60 /65
Chef de projets MOE	NS	NS	50 /60	60 /65
Responsable R&D	NS	NS	55 /63	63 /70
Directeur technique	NS	NS	NS	60 /75
Architecte/Urbaniste	NS	NS	55 /60	60 /75
Integrateur web	28 /35	35 /38	38 /42	42 /45
Webmaster	25 /32	32 /35	35 /40	40 /45

En k€ bruts, salaire moyen, en île de France –
source : Hays « informatique & télécoms un marché stable »

Le cabinet Robert Half a édité son étude sur les rémunérations 2016. Pour les développeurs, le salaire moyen brut est estimé à 42 – 57 k€ pour 3 – 5 ans d'expérience. La fourchette est plus large que dans l'étude précédente, avec grosso modo le même niveau de départ. Cependant, un développeur mobile (avec la même expérience, ce qui est difficile à trouver), la fourchette est moins importante et le salaire moyen moindre, ce qui nous paraît en décalage avec les contraintes actuelles.

fonction	expérience (années)	salaire (k€ / annuel)	évolution 2014/2015
Ingénieur dév.	3-5	42-57	+4,2 %
(PHP, Java, C++, .net)	5-10	56-64	+3,9%
	10-15	64-72	+4,4 %
Ingénieur dév. Mobile	3-5	41-51	+4,2 %
	5-15	51-60	+4%
architecte	5-10	62-77	+3,3 %
	10-15	78-94	+4,1%

On constate aussi que le directeur des données (CDO) dépasse les 100 k€, mais c'est un profil très spécifique et exigeant sur les compétences. Par contre, nous sommes étonnés par le faible niveau du Data Scientist, de 35 à 65 k€.

Le terme intégrateur HTML est toujours présent, mais il ne correspond plus réellement à un profil clair, on parlera plutôt de développeur web, voire, d'intégrateur web. Là, les entrants connaissent des salaires très bas, à partir de 25 k€.

5e édition

Congrès & Expo

BIGDATA

Turning the data deluge into decisions

Rendez-vous les
7 & 8 mars 2016
Palais des Congrès de Paris

Les supers pouvoirs du Big Data !

Participez à ***l'événement leader du Big Data*** en France

- 10 000 participants
- 150 exposants
- 100 experts à la tribune
- 5 salles de conférences

Réservez dès à présent vos ***7 & 8 mars 2016*** et préparez ***l'avenir*** de votre entreprise avec ***Big Data Paris !***

Votre badge gratuit sur www.bigdataparis.com

by CeBIT in Corporations we Trust

- IDE : 83 %
- Framework : 61 %
- Composant : 29 %
- Base de données : 75 %
- Base NoSQL : 21 %
- Outils et méthodes agiles : 32 %
- Référentiel de code : 47 %
- Un outil Cloud : 24 % (dont 10 % pour un service de build en ligne)
- La virtualisation est assez fortement utilisée au quotidien (25 %). Et vous êtes 14 % à utiliser un outil dédié au refactoring.

Les tests ne sont pas oubliés ! Vous êtes 35 % à déclarer utiliser (au moins) un outil dédié aux tests. Parmi les IDE phares, Visual Studio est largement en tête de liste avec 42 %, Eclipse 30 %, Android Studio 20 %, XCode arrive tout de même à 13 %, tandis que NetBeans est loin derrière 10 %.

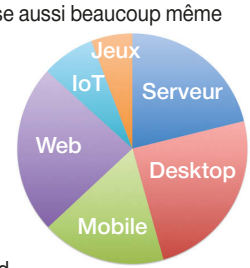
Des langages nombreux mais pas tant que cela ! (tableau ci-dessous)

Nous vous parlons réellement de l'index TIOBE sur la popularité des langages. Qu'en est-il de l'enquête ? Mais vous regardez aussi d'autres langages que votre langage principal actuel pour les prochains. Ainsi, on constate que Swift vous intréresse (15 %), Python (21 %), Scala / WLangage (6,5 %). Mais les langages « classiques » continuent à être largement devant pour les prochains mois (C / C++, Java, JavaScrit, PHP, C#). Par contre, Objective-C ne vous passionne pas, ni les langages fonctionnels...

Des développements très diversifiés

On constate aussi une grande diversité des projets des développeurs : voir graphique.

Le Maker vous intéresse aussi beaucoup même si vous êtes 43 % à ne jamais en avoir fait, vous êtes 43 % à en faire souvent ou parfois...



La mobilité

Sur la mobilité, Android est la plateforme principale mais Windows Phone arrive 2e devant une courte tête devant iOS. C'est l'illustration du marché français où Apple est en retrait. Le développement natif demeure la règle (41 %).

Sur ce dernier point, notons effectivement l'excellente tenue du développement natif comme nous l'avons dit (41 %). Les développements multiplateformes ne sont pas oubliés.

Série

MR ROBOT : une vraie série de hacking

J'ai mis quelques semaines avant de regarder la série MR ROBOT. Tout d'abord, sans forcément être très concentré, mais très rapidement, le premier épisode accroche. Il prend à la gorge et cette impression ne te lâche plus.

L'histoire est celle d'un hacker et nerd (il n'aime pas vraiment les autres et encore moins la foule). Il bosse dans une boîte de sécurité qui s'occupe d'un énorme client omnipotent. Piratant les données de personnes pour les dénoncer ou pour son plaisir, il est parano et surveille tout le monde. Suis-je suivi ou pas ? C'est alors qu'il est abordé par MR ROBOT après avoir déjoué une violente attaque contre les serveurs d'Evil Corp qui va le faire bousculer dans la fsociety... Le but final : anéantir Evil

Corp et l'économie...

Mais rien n'est réellement simple. Un plan existe (celui des Cylons ?), mais lequel exactement ? On sent que la santé mentale de notre hacker Elliot n'est pas stable. Et c'est encore pire quand on découvre la réalité dans les derniers épisodes. De nombreuses questions se posent sur sa mémoire et ses motivations que l'on comprend mal finalement.

La révélation (désolé pas de spoiler) nous met une sacrée claque, mais finalement, on commence à comprendre certaines choses, surtout, quand on regarde une 2e fois les épisodes.

Nous avons adoré l'ambiance, la qualité de la réalisation, les acteurs. Et des personnages que l'on pourrait croire sans intérêts se révèlent sans le moindre scrupule. L'un des pires est Tyrell Wellick, mais sa femme, Joanna, rivalise largement...

Plusieurs histoires secondaires occupent parfois trop certains épisodes même si on comprend leurs influences dans la trame centrale.

Contrairement à ce que l'on pourrait penser, vous êtes 15 % à utiliser Cordova et à peine 6 % à vous tourner sur Xamarin. Et vous êtes encore moins nombreux à utiliser Appcelerator...

Je cloude moi non plus !

Le Cloud Computing est partout ou presque. Vous êtes 57 % à utiliser au moins un service Cloud mais pas forcément pour le développement. Les usages sont très divers :

- Stockage 37 %
- Hébergement d'un site Web 26 %
- Service SaaS : 20 %
- Base de données : 17 %
- Référentiel de code : 16 %

Le PaaS reste très minoritaire dans votre quotidien (6,5 %). Même éclatement dans le choix du fournisseur. Azure est cité à 18 % mais les choix sont très diverses (AWS, Heroky, CloudFoundry, Bluemix, Google, etc.). Cependant, dans les prochains mois, vous êtes 38 % à vous intéresser plus sérieusement au Cloud Computing...

Et demain ?

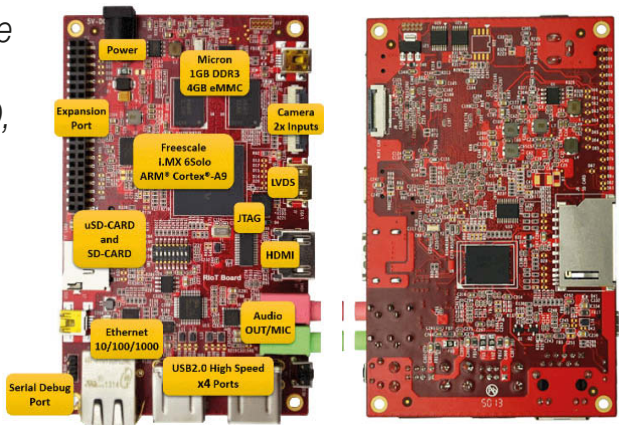
Là encore, nous retrouvons plusieurs technologies tendances ou porteuses comme le Cloud, les IoT et la 3D.

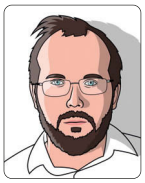
html / css	SQL	JavaScript	PHP	Java C#	C C++	VB	Python	Delphi	Ruby Groovy WLangage
63 %	60 %	51 %	37 %	32 %	31 %	21 %	16 %	11 %	5 %

cloud	IoT / maker	wearable	3D	autre
38 %	40 %	13 %	19 %	20 %

RIoTBoard : une plateforme très complète

Peu connue en France, la carte RIoTBoard est une plateforme soutenue par Freescale qui fournit d'ailleurs le processeur Arm Cortex-A9. Ce processeur est assez puissant pour la 2D, 3D, la vidéo 1080p. Que faire avec cette carte ?





François Tonic
Programmez!

La carte est très complète en composants et interfaces :

- Processeur ARM ;
- Mémoire vive 1 Go ;
- Connecteurs cartes flash ;
- Ports USB ;
- HDMI, audio, camera, ethernet, display, etc.

La fabrication est très propre. La carte ressemble à une Raspberry Pi ou une Intel Galileo. La carte cible tout d'abord les objets mobile, les mini-PC, les petites consoles, les IoT en général, mais vous pouvez aussi l'utiliser pour prototyper rapidement des objets. En termes de puissance, on peut la comparer à la Raspberry Pi 2 la carte propose toutefois bien plus de fonctionnalités de base et d'interfaces. Pour démarrer rapidement avec la carte, element14 propose plusieurs modules : écran 7 pouces, caméra, carte BEE, GPS, Bluetooth, capteur d'environnement.

La partie logicielle

Deux systèmes sont supportés par défaut par la carte : Linux et Android. Pour Linux, vous trouverez des images Ubuntu, Yocto.
Pour Android, les versions 4.3 et 4.4 sont officiellement disponibles. Utilisez les images officielles.
Mais par défaut, la partie logicielle est limitée, même si vous allez utiliser les outils de développement classiques à chaque système.

Le GPIO

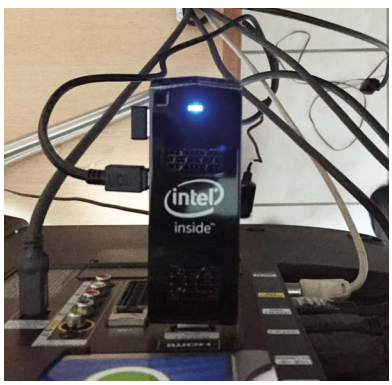
L'interface d'extension comporte 40 broches : 3,3 / 5v, GND, GPIO, UART, I2C. L'utilisation du GPIO se fait de la même manière qu'avec Arduino ou une Pi 2. On connecte les capteurs sur les broches adéquates. Ensuite il faut coder !
Pour simplifier le travail et pouvoir utiliser facilement des modules Raspberry Pi, Arduino et MikroBUS, un adaptateur est disponible (RIoT Adapter). Ce shield se connecte directement sur le GPIO. Vous vous ouvrez ainsi tout l'écosystème maker. Nous vous conseillons vivement cet accessoire.

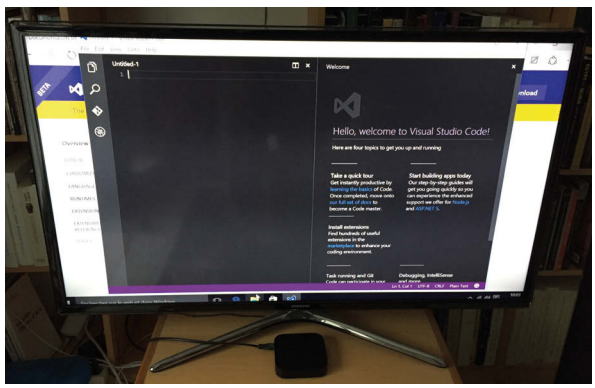
Puissant et plus expert

A l'usage, cela ressemble un peu au Pi 2 pour la programmation, le branchement des capteurs. La RIoTboard en a sous la puce mais elle s'avère un peu plus complexe à dompter pour un Maker ; un développeur sera moins perdu. Mais honnêtement, cette carte n'est pas plus compliquée qu'une Pi 2 et les distributions Linux fonctionnent très bien.
Et le support d'Android est un plus indéniable pour construire des bornes et box indépendantes.
Le point négatif est le manque de tutoriaux orientés capteurs / montages et une communauté francophone faible. Le tarif de la carte est d'environ 70 € H.T. et env. 15 € H.T. pour l'adaptateur Pi / Arduino / Mikrobus.
Site officiel : http://riotboard.org

INTEL COMPUTE STICK : UN PC COMPLET SUR UNE (GROSSE) CLÉ

C'est une nouvelle manière de concevoir un ordinateur. L'objet ressemble à une grosse clé USB qui aurait pris du volume. Mais au lieu d'un connecteur USB, nous avons une interface HDMI. Il s'agit d'enfoncer la clé dans un port HDMI d'une télé ou d'un moniteur. Mais, il nécessite obligatoirement une alimentation externe pour fonctionner. Le système embarque un processeur Atom 4 cœurs, un port USB 2, un connecteur Micro-SD, un bouton de démarrage, bluetooth 4 et le WiFi. La version Windows 10 possède 2 Go de mémoire et 32 Go de stockage, la version Linux se limite à 1 Go et 8 Go de stockage. Si vous ne voulez pas vous encombrer d'un ordinateur ou l'utiliser en poste de secours, la solution est satisfaisante, surtout si vos besoins sont limités ou peu exigeants. Vous pouvez parfaitement créer votre poste de développement de secours dessus mais n'installez pas des IDE ultra complets. Visual Studio Code s'installe sans problème par exemple. Attention tout de même à l'espace de stockage. Le stick a tendance à chauffer (particulièrement en version Windows) et le câble USB pour l'alimentation, livré dans la boite, est un peu court. L'accès à Battle.net est proposé par défaut. La finition est correcte mais un petit effort ne serait pas inutile, tout comme sur le design général qui mériterait d'être plus arrondi. Tarif officiel : env. 150 €.





Le téléphone modulaire : l'avenir du smartphone ?

Obsolescence programmée, réparation limitée et difficile, renouvellement régulier, le marché des smartphones est aujourd'hui un marché mature avec des pays en pleine extension. La stratégie varie énormément selon le constructeur : forte marge, marge très faible, gamme très large, gamme réduite, etc.

Fairphone

PuzzlePhone

François Tonic
Programmez!

Depuis plusieurs années, on parle du smartphone modulaire, à réparer soi-même, à « construire » soi-même. Google travaille sur le sujet avec le projet Ara. Ara est la promesse de pouvoir choisir les modules que l'on veut, et de construire son téléphone sur un socle matériel. Mais le projet a patiné sur plusieurs écueils : stabilité du système (il faut gérer les multiples modules, les changements à chaud, etc.) et les problèmes techniques comme la bonne adhérence des modules sur la plateforme technique en cas de chute... Depuis septembre 2015, les équipes Ara se font très discrètes. En août dernier, Google avait décalé la sortie réelle du projet à 2016, sans plus de précisions.

Fairphone 2

Fairphone est une approche moins radicale, mais qui se veut éthique et durable. Fairphone est un smartphone facilement réparable sur lequel on peut changer l'écran, la coque, le module photo, la partie audio, la batterie, très facilement et acheter, chez le constructeur, les nouveaux modules. Cela change des smartphones sur lesquels, on ne peut quasiment rien changer soi-même.
Fairphone a dévoilé la version 2 de son mobile avec un écran 5 pouces full HD, Android 5.1, une double SIM, le support des technologies sans-fil, le tout avec un processeur 4 cœurs.
L'objectif est de réduire l'usage de minerais rares (même s'ils sont indispensables à l'industrie des technologies). Le concept est séduisant et montre ce que Ara pourrait être.
Le constructeur se veut aussi transparent sur les coûts et la marge d'exploitation sur chaque modèle vendu. Le Fairphone 2 est vendu au prix public de 525 € (ce qui est assez élevé pour un modèle Android). La fabrication, les matériaux pèsent 340 €, l'investissement 33.
L'objectif est ambitieux : produire et vendre 140 000 terminaux par an. Condition pour pérenniser le projet. Le volume est très faible par rapport aux autres constructeurs, ce qui pénalise Fairphone et augmente de facto les coûts de fabrication.
Le constructeur met aussi en avant sa volonté de mieux contrôler l'origine des matières premières et soutenir les économies locales ; par exemple, bannir les minerais provenant de zones de guerres que l'on appelle les minerais du sang.
L'initiative est intéressante même si le prix est élevé, l'autonomie bonne, mais sans plus (environ 1 journée). Sur la partie design, vous ne retrouverez pas les finesses des modèles hauts de gamme.
Site : fairphone.com

PuzzlePhone

Autre mobile modulaire, le PuzzlePhone. Il se veut lui aussi modulaire (un peu) et facile à adapter/réparer. À la base, il repose sur une plateforme matérielle (le cerveau contenant CPU, GPU, mémoire, etc.) sur laquelle se rajoutent l'écran et la batterie. Le constructeur met en avant son côté ouvert et la possibilité de créer, par des tiers, des modules additionnels, mais les possibilités se limitent aux trois composants/modules. Et PuzzlePhone veut même encourager les créations pour son mobile. Et le développeur n'est pas oublié, même si aucun SDK n'est disponible publiquement. Le PuzzlePhone n'est pas attendu avant septembre/octobre pour un tarif de 299 € (à partir de).
Site : puzzlephone.com

Rephone

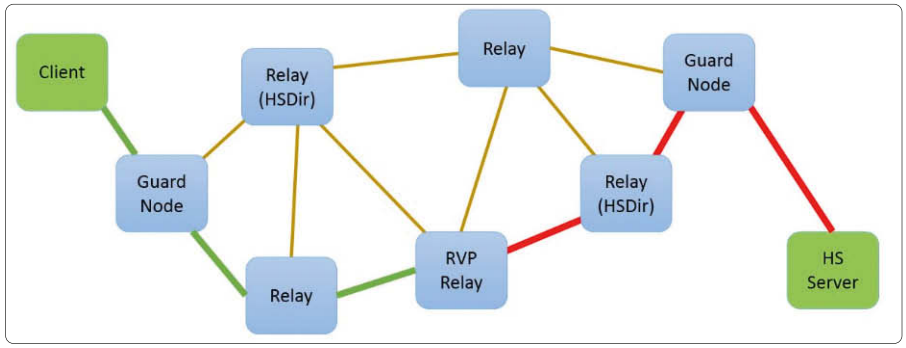
Rephone : un mobile fait par des makers pour des makers !

Ce projet que nous suivons depuis plusieurs mois est l'un des plus intéressants pour les makers et développeurs : Rephone. Initié et soutenu par Seeed Studio, bien connu des makers pour les composants et capteurs en tous genres, Rephone est un mini-téléphone mobile modulaire, open source et à monter soi-même ! Il repose sur un environnement technique bien connu : Arduino, Lua, JavaScript et l'open hardware ! Il ne prétend pas être un smartphone aussi puissant qu'un Ara ou Fairphone, mais il est bien plus amusant et personnalisable. De multiples modules seront disponibles !
Le kit de démarrage est proposé à 59 € !
Livraison prévue courant janvier/février.
Site : http://www.seeed.cc/rephone/

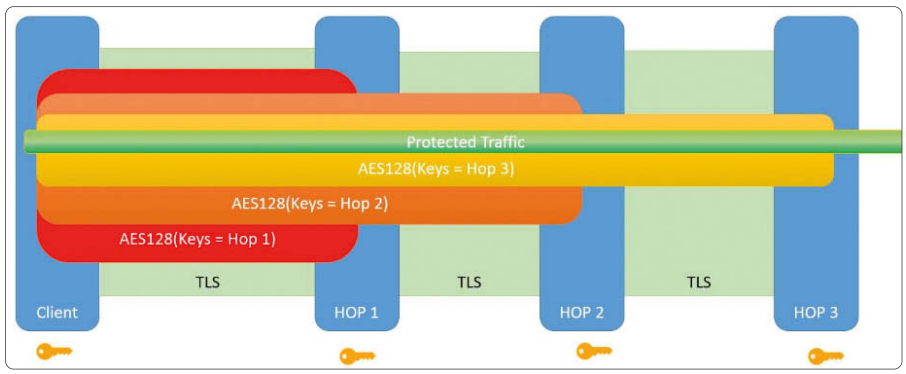
Parfois le crime paie : incursion dans le Darknet

Pour la première fois la célèbre conférence de sécurité RSA faisait étape au Moyen-Orient. C'est à Abu Dhabi que Greg Jones, spécialiste en cyber assurance, a embarqué son audience pour un voyage dans le Darknet, avec un focus sur la cybercriminalité.

Véronique Loquet
Fondatrice de l'agence RP AL'X Communication.
Spécialiste de l'Open Source et de la sécurité.
Twitter @vloquet



Voie de communication simplifiée pour un service caché de tor
7 relais sont présentés pour plus de simplicité, il y a actuellement près de 6500 relais opérationnels



Vue simplifiée de la protection du trafic dans Tor. Clés RSA publiques pour l'authentification et les identités nœud. Clés symétriques AES pour la protection du trafic.

Déjà le terme Darknet prête à confusion et laisse à penser qu'une fois son IP planquée chaque internaute se livrerait à un sombre trafic. Or cette face cachée de l'Internet est un peu comme les catacombes, un univers souterrain qui n'a pas les réseaux criminels pour uniques visiteurs. En y plongeant, vous pourriez y trouver votre voisin soucieux de sa vie privée, les forces de l'ordre, des lanceurs d'alerte, hacktivistes, journalistes, dissidents politiques... Bref un tas de gens désireux d'évoluer dans un espace anonymisé, qui donne aussi accès à une multitude de sources non indexées par les traditionnels moteurs de recherche.
L'approche de Greg Jones consiste à mieux comprendre les menaces criminelles de cet univers underground. Les réseaux anonymes et cachés du Darknet permettent l'hébergement de contenus et services où il est extrêmement difficile d'identifier les serveurs en cours d'exécution, et par là même, de prendre toute action légale à l'encontre de leurs utilisateurs.
Il n'existe pas un Darknet mais une pleine mosaïque, avec une économie parallèle élaborée et résiliente.
Les caractéristiques de connexion doivent au minimum permettre de garantir que l'adresse IP sous-jacente, le cas échéant, ne puisse être lue, que les communications soient à la fois authentifiées et chiffrées, et l'architecture décentralisée. Au mieux les attributs auront des options de latence variables, des chemins de trafic asymétriques et un réseau indépendant.

Actuellement on compte quatre principaux chemins d'entrée vers les réseaux du Darknet.
Freenet, l'un des premiers est apparu au milieu des années 90 ; toujours très utilisé et basé sur du P2P, il est axé sur le stockage de contenu distribué. Comparable à BitTorrent, mais ses données stockées sont chiffrées.
GNUnet apparaît en 2000, également basé sur le P2P, il se concentre sur l'hébergement de contenus fichiers au travers de circuits arbitraires disponibles pour la messagerie, voix, etc.

I2p (*Invisible Internet Project*) arrivé en 2003, basé sur le P2P, c'est un réseau commuté par paquets. I2P est une surcouche réseau d'anonymisation, comme pour les VPN il exploite la tunnelisation pour fournir un « réseau dans le réseau ».
Tor (*Tor onion router*) a été conçu dans les années 90 par un laboratoire de l'armée américaine, il fait son apparition dans le grand public en 2004. Il se compose de routeurs organisés en nœuds, pour transmettre des flux TCP de manière anonyme. C'est un logiciel libre distribué sous licence BSD.
Très efficace, Tor est aujourd'hui le plus populaire d'entre eux, tous ses services cachés sont accessibles en utilisant un .onion, comme par exemple :

- silkroad6ownowfk.onion – Silk Road (v2) – disparu en octobre 2014
- facebookcorewwwi.onion – Facebook – Lancé voici un an.
- 3g2upl4pq6kufc4m.onion – Duck Duck Go, un moteur de recherche

Tout nœud Tor, y compris pour les clients, peut publier un descriptif de service caché auquel tout autre noeud Tor peut se connecter, indépendamment de la topologie IP sous-jacente. Le principe est que chaque communication passe par une combinaison de trois serveurs, chacun d'entre eux n'identifiant que l'adresse IP de l'étape précédente. La principale propriété d'un service caché est qu'il est non trivial de déterminer où le service est hébergé.
Pour une parfaite confidentialité, en plus de Tor les utilisateurs peuvent utiliser un VPN. La navigation sera assurée à l'aide de moteur de recherche comme Grams (URL-*grams7enufi7jmdl.onion)* proche de l'utilisation de Google, celui de Tor, *Onion city* (http://onion.city/) ou encore via wikis.

Crime traditionnel versus Dark économie

Dans l'univers du crime sans Internet, il est nécessaire de connaître les « bonnes personnes », de les rencontrer avec ce que cela implique de danger pour l'intégrité physique et les chances d'être repéré, et suppose en outre d'avoir du cash disponible. Avec la criminalité en réseau, un catalogue de contacts spéci-

fiques est disponible en ligne, elle offre aussi la possibilité d'obtenir un feedback sur la réputation du contact. Quant aux transactions, elles s'effectuent au choix de la crypto-monnaie, parmi des centaines similaires au Bitcoin.
De nombreux forums illégaux prospèrent sur le Darknet avec une large gamme d'opportunités pour les participants. Les places de marchés online permettent aux fournisseurs d'y faire leur publicité, on compte environ 4000 fournisseurs. La plupart exercent un filtrage nécessitant un droit d'accès parfois payant, d'autres l'accord de l'administrateur ou le pass d'un parrain du milieu... L'offre disponible n'a de limite que l'imagination des malfaiteurs : vente de stupéfiants, blanchiment d'argent, services de hacking, kit de Phishing, vente d'armes, fraude administrative, accès à des sites gouvernementaux, vente de véhicules volés, location de Botnets, dispositifs de piratage, vente d'exploits, tueurs à gage... Avec un chiffre d'affaires stable, d'environ 20 millions de dollars US par mois, principalement en Bitcoin. Les plus grandes places de marchés sont SR, Atlantis, Sheep, Evolution et Agora. La plus importante actuellement serait Abraxas. Certains vendeurs ont leur propre site Web, d'autres négocient sur les forums, via emails ou messageries instantanées. Le plus souvent les délits sont réalisés au delà des frontières géographiques du pays où l'argent du crime converge.
La tendance à venir est au développement des marchés décentralisés.
Un forum de piratage très actif (Hell) a récemment été stoppé lorsque l'opérateur (Ping) a été arrêté pour piratage et accusations de fraude. La fermeture de ces forums et marchés noirs s'effectue de trois manières :

- Ils sont saisis et stoppés par les forces de l'ordre ;
- Ils sont piratés entrainant la perte de tout leur stock de Bitcoins ;
- Leurs opérateurs les ferment et s'enfuient avec les Bitcoins .

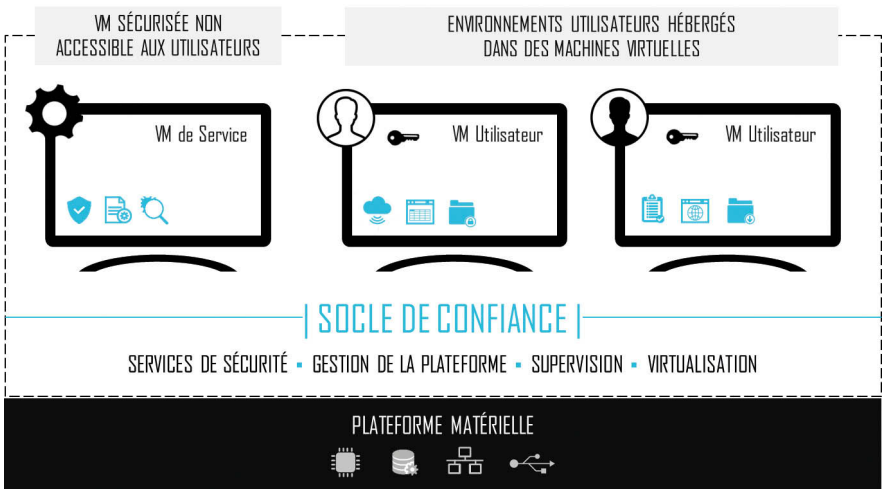
Quid de la lutte anti-criminalité ?

Pas simple car la professionnalisation des groupes criminels fait face à une certaine inertie ; des moyens juridiques qui ne répondent pas toujours aux situations et aux délais compatibles de l'enquête. Les preuves numériques sont indispensables pour faire avancer l'enquête, mais hautement volatiles (historiques de navigation, stockage de données, logs de connexion, adresses IP...). Généralement les victimes utilisatrices de services illicites ne portent pas plainte. En France, le code pénal précise que les faits de l'infraction doivent être commis sur son territoire pour que la loi s'applique. Cette criminalité transfrontalière a fait évoluer les dispositions et le législateur ; malgré une plus large coopération internationale entre forces de l'ordre, il reste de nombreux obstacles à l'aboutissement d'une investigation. L'infiltration par les enquêteurs habilités à utiliser de fausses identités est encadrée et ne concerne qu'une liste définie d'infractions au code pénal. De même, l'investigation sous pseudonyme n'autorise pas les enquêteurs à provoquer l'infraction. Les Honeypots servent de leurres et peuvent permettre d'observer un mode opératoire, mais en vertu du principe de loyauté d'obtention de la preuve pénale, les éléments recueillis via ce procédé seront irrecevables devant la cour. L'officier de police judiciaire peut procéder à l'interception des communications, mais si les connexions se font depuis une plateforme à l'étranger, l'information n'est pas directement accessible via l'opérateur. Si le suspect est basé en France, il pourra cependant capturer les données à distance, perquisitionner et saisir des disques durs, périphériques de stockage et terminaux. Pour les données stockées situées dans un Cloud hors du territoire, la coopération internationale peut s'engager, sous réserve de la bonne volonté et des moyens des pays concernés.
Des opérations d'envergure ont conclu à la fermeture de sites illégaux, comme celle de *Onymous* menée par Europol et le FBI, qui a suspendu plus de 400 sites de ventes d'armes et de stupéfiants, cependant elles n'entraînent pas systématiquement de poursuites judiciaires. Ross Ulbrecht, le fondateur de Silkroad condamné à perpétuité en 2015, fait figure d'exception notamment pour l'extrême sévérité de la sentence. Ces sites ressuscitent rapidement. En outre, certains pays voient un intérêt à protéger ces réseaux criminels en leur offrant un paradis numérique à l'instar des paradis fiscaux. Une garantie d'impunité qui renforce l'émergence de nouveaux marchés noirs et laisse un boulevard pour des affaires criminelles organisées et très lucratives. ⊞

La France a développé / développe des projets sécurisés

Alors que des députés proposent de créer un système d'exploitation français et souverain, une petite recherche ici et là permet de découvrir rapidement que la France avait / a plusieurs projets sécurisés :

- Hyperviseur Polyxène : hyperviseur hautement sécurisé (certification EAL 5) à plusieurs niveaux, il a été développé par Berlin Technologie dans le cadre d'un projet de la DGA (délégation générale pour l'armement). Il a été déployé en projet pilote en 2014.
- SINAPSE : dès 2004, la DGA a lancé la conception d'un système sécurisé. SINAPSE signifie Solution Informatique à Noyau Avancé Pour une Sureté Elevée. Berlin Technologies était déjà au développement. Une v2 a été lancée en 2014.



La France suit donc de près d'autres pays qui se sont équipés de projets identiques notamment pour l'administration ou les armées : Etats-Unis, Corée du Nord, Russie, Chine, l'Inde.

Google passe à OpenJDK pour Android N : perspectives pour les développeurs

Durant les vacances de Noël, Google a confirmé sa volonté de remplacer son implémentation des APIs Java présentes au sein d'Android par l'implémentation open source d'OpenJDK. Ainsi, Android N, la prochaine version de l'OS mobile de Google, s'appuiera sur OpenJDK. Il n'en fallait pas plus pour agiter la sphère Java et laisser place à tout un tas de spéculations sur les conséquences de cette annonce que ce soit d'un point de vue juridique ou technique. Dans cet article, nous nous plaçons principalement du point de vue du développeur en cherchant à déterminer les perspectives offertes par cette annonce.

Sylvain SAUREL
Ingénieur d'Etudes Java / Android
sylvain.saurel@gmail.com – www.all4android.net

Tout est parti d'un commit réalisé par les équipes de Google sur la base de code d'Android il y a un peu plus d'un mois. Toujours en quête de nouveautés concernant l'avenir de la plateforme, des développeurs l'ont ainsi repéré avant de donner plus de visibilité à son contenu. Le commit réalisé concerne la modification de près de 9000 fichiers et explique clairement qu'il s'agit de l'import initial des fichiers d'OpenJDK sur la base de code d'Android N. Devant l'ampleur prise par les évènements, Google a ensuite pris la peine de réagir publiquement en confirmant qu'il allait remplacer sa propre implémentation des APIs Java d'Android par l'implémentation open source d'OpenJDK.

Jusqu'à présent, le SDK Android de Google était basé sur Apache Harmony une implémentation open source poussée par la fondation Apache se voulant compatible avec la licence Java originellement détenue par Sun. Google était bien rentré en contact avec Sun à l'époque pour régler les problèmes de licence liés à cette utilisation mais n'était pas parvenu à trouver d'accord. Le rachat de Sun par Oracle en Janvier 2010 ayant ensuite ouvert la voix au fameux procès opposant Oracle et Google sur le caractère copyrightable ou non des APIs Java. Après plusieurs rebondissements donnant raison à l'un ou l'autre des deux camps, l'affaire n'est toujours pas réglée en ce début d'année 2016. Certains voient donc dans le choix de Google d'opter pour un passage à OpenJDK un accord futur ou peut-être déjà conclu avec Oracle. Cependant, le fait qu'Oracle ait préféré ne pas s'exprimer sur le sujet ne permet pas d'en dire beaucoup plus sur le volet juridique de cette annonce.

Perspectives pour les développeurs

Aussi récente que soit cette annonce, nous pouvons d'ores et déjà nous projeter et voir quelles sont les perspectives pour les développeurs d'un point de vue technique. Le passage à OpenJDK doit permettre de réduire l'écart existant entre le Java classique que nous connaissons et le Java des applications Android. En ce sens, Android N devrait rendre la vie des développeurs Java plus facile puisqu'ils pourront employer les classes Java standards de la même manière que sur un environnement Java classique.

A l'heure actuelle, de nombreux développeurs rencontrent des problèmes en tentant d'utiliser des bibliothèques de code du monde Java au sein de leurs applications Android avec des plantages nécessitant des contournements voire une incompatibilité complète. Cela tient au fait que l'implémentation des APIs Java d'Android diffère du Java standard avec un certain nombre de classes en moins, des méthodes absentes ou des comportements quelquefois différents. L'intégration d'OpenJDK au sein d'Android rendra du même coup fonctionnelles ces bibliothèques.

La plus grande problématique pour les développeurs restera toutefois la compatibilité. En effet, le nombre de classes implémentant les APIs de Java est très important et les packages *java.** et *javax.** contiennent beaucoup de code. Il y aura aussi forcément des ajustements à faire puisque les APIs Java d'Android peuvent avoir certains comportements différents auxquels les applications ont du s'adapter. Le passage à OpenJDK pourrait donc se faire avec quelques régressions. Il y aura ainsi surement besoin de recourir aux classiques exceptions au sein du code à l'aide de la constante *Build.VERSION.SDK_INT* pour gérer les versions pré Android N qui s'appuieront encore sur l'implémentation des APIs Java de Google.

Google ayant déjà annoncé qu'il lui sera nécessaire d'adapter la base de code OpenJDK pour son intégration au sein du SDK Android, il faut espérer que le géant de Mountain View fasse un important effort de documentation pour donner aux développeurs une vision claire sur les packages supportés en l'état dans Android N, ceux qui pourront être ajoutés dans le futur et enfin les packages qui ne seront pas supportés puisqu'ils peuvent supporter OpenJDK en ne prenant qu'un sous-ensemble des fonctionnalités de Java.

Java 8

Avec son implémentation basée sur Harmony, Google était coincé au niveau de Java 6 jusqu'à présent. Cette version a près de 10 ans ce qui représente une éternité dans le monde de l'informatique d'autant plus que le monde Java a connu une réelle révolution il y a de cela 2 ans avec l'arrivée de Java 8. En s'appuyant sur l'implémentation open source d'OpenJDK, les développeurs auront désormais accès à un grand nombre de nouveautés parmi lesquelles :

- Les Streams ;
- Les Lambdas ;
- Les Interfaces Fonctionnelles ;
- L'API Date and Time.

Cette liste n'est bien entendue pas exhaustive et les futures évolutions de Java 9 seraient également accessibles de manière assez rapide.

Néanmoins, et compte tenu du rythme d'adoption des nouvelles versions d'Android, il faudrait sûrement attendre près de 4 ans avant que la nouvelle version d'Android basée sur OpenJDK puisse être répandue sur au moins 90% du parc Android. Pour combler le gap durant cette période de transition, un projet comme Retrolambda pourrait gagner en popularité auprès des développeurs Android.

Quand on voit l'effet stimulant qu'aura eu Java 8 sur la plateforme Java, on peut espérer un effet similaire autour d'Android avec un afflux massif de développeurs Java souhaitant tirer partie des nouvelles possibilités offertes. En effet, il sera possible d'écrire du code de manière plus moderne tout en tirant partie des nombreuses bibliothèques tierces du monde Java ce qui donnera du même coup de meilleures applications.

JavaFX

Une autre grande question qu'amène ce rapprochement concerne JavaFX. En effet, JavaFX bénéficie également d'une implémentation open source nommée OpenJFX qui est directement alignée sur le code d'OpenJDK. Le framework est désormais la solution la mieux intégrée à OpenJDK pour réaliser des IHMs en Java. Actuellement, des initiatives existent pour porter JavaFX sur Android focalisées sur 2 pré-requis :

- S'assurer que le code source d'OpenJFX n'utilise pas des APIs de Java SE qui ne sont pas disponibles sur Android et écrire du code de contournement lorsque cela ne peut être évité ;
- Porter le code de rendu natif des composants sur la plateforme Android.

Le passage à OpenJDK règlera de facto le premier point en laissant le soin aux équipes de se concentrer sur le rendu natif sous Android. Un support complet de JavaFX pourrait s'avérer stratégique pour unifier le développement d'IHMs entre le desktop et Android. Bien entendu, il s'agit sur ce dernier point de prospective mais Google pourrait également y trouver son intérêt en se dégageant plus de temps pour développer des services spécifiques sous Android et en s'affranchissant un peu plus du côté IHM qui serait porté par OpenJDK et OpenJFX.

OpenJDK

En sus de son passage à OpenJDK à partir d'Android N, Google a également annoncé son intention de travailler plus en profondeur avec la communauté OpenJDK via une plus grande contribution afin de pouvoir jouer un rôle plus important dans l'avenir de cette implémentation des APIs Java. Voir Google contribuer à cette implémentation open source dirigée par Oracle serait également un signe positif qui rassurerait complètement les développeurs Android et Java lassés de la lutte que se livrent ces deux géants pour des histoires de gros sous. On peut également y voir le signe que Google cherche à renforcer sa position en se rapprochant d'Oracle au moment même où Microsoft cherche à implanter ses services dans le monde Android via différents accords avec Cyanogen qui édite une ROM Android custom très populaire.

Il reste également à espérer qu'Oracle et Google trouveront finalement un accord et qu'il sera reconnu que des APIs ne peuvent être copyrightées sous peine de voir un précédent s'installer dans le monde du développement avec une emprise encore plus forte des géants du Web proposant des APIs et services aux développeurs.

Android, projet Sky et Dart

Au printemps dernier, on évoquait le projet Sky (appelé désormais Flutter). Il s'agissait de créer un framework technique basé sur le langage Dart et destiné aux applications mobiles. Immédiatement, la rumeur d'un remplacement de Java par Dart a surgi. Après quelques mois, il apparaît que cette possibilité s'éloigne car cela signifierait des changements radicaux dans le système même si Apple a montré la possibilité de changer de langage avec Swift.

Site : https://flutter.io

La rédaction

Conclusion

Pour le moment, on ne peut qu'envisager les impacts réels qu'aura à l'avenir le passage d'Android à OpenJDK. D'un point de vue juridique, il est à espérer que la situation s'arrange entre Oracle et Google pour le bien de l'écosystème Java. D'un point de vue technique, les développeurs d'applications d'Android n'ont pour le moment pas de grandes craintes à avoir puisque les problèmes de compatibilité éventuels ne devraient pas être légions. Cependant, il sera primordial de tester en avance ses applications sur la developer preview d'Android N afin d'anticiper au mieux d'éventuels changements à réaliser au sein de son code. Pour le reste, on ne peut que se réjouir de la décision prise par Google de basculer vers l'implémentation open source d'OpenJDK.

SQL, NoSQL : Comment choisir sa base de données ?

Aujourd'hui, les développeurs ont la possibilité de choisir leur base de données. Le temps où la seule option possible pour créer son application Web était MySQL est révolu. Vous allez découvrir d'autres bases de données qui vous permettront de répondre de manière plus efficace à vos problématiques.

Loïc Guillois /
Développeur chez LaFourchette.com

Un peu de théorie

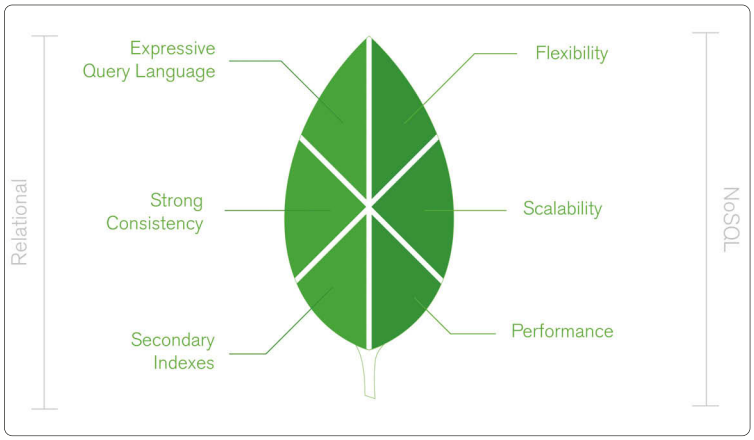
Ce qui définit les bases SQL que l'on connaît depuis des dizaines d'années, ce sont leurs caractéristiques ACID. Un SGBD permet de réaliser des transactions atomiques, cohérentes, isolées, et durables. Ces avantages sont aussi des inconvénients en termes de performance en introduisant des goulots d'étranglement dans l'architecture applicative, notamment à cause des verrous. C'est pourquoi depuis quelques années, un nouveau modèle émerge pour se différencier de ce modèle relationnel en allant jusqu'à abandonner le langage SQL.
Ainsi, certaines bases abandonneront la durabilité : c'est le cas des bases en mémoire vive. D'autres encore abandonneront les transactions rendant ainsi impossibles les opérations atomiques et les retours arrière classiquement utilisés. Les propriétés ACID laissent donc place à trois autres caractéristiques (CAP) auxquelles les bases NoSQL tentent de répondre :

- Cohérence (Consistency): tous les nœuds d'un cluster disposent des mêmes données au même moment ;
- Disponibilité (Availability) : à tout moment, une requête reçoit une réponse ;
- Tolérance au partitionnement (Partition Tolerance) : un problème sur un nœud du cluster ne doit pas empêcher le fonctionnement du cluster dans son intégralité.

Au-delà des principes d'architecture, ce qui va différencier les bases SQL des bases NoSQL : c'est le format de stockage. En SQL, on travaille avec des tables contenant des lignes. Ces données sont structurées selon un schéma contraint et des relations sont possibles entre les différentes lignes. Une base NoSQL s'affranchira de cette approche et pourra stocker les données sous différentes formes : document (XML, JSON...), objet, clef/valeur, colonnes ou encore graphe par exemple.
Comme nous allons le voir, ce ne sont plus le langage ni le framework utilisés qui déterminent la base de données que l'on utilise. On choisit la base de données qui correspond le mieux à notre besoin.

Redis : zoom sur une base clef/valeur

Redis a pour caractéristique principale de proposer un mécanisme de stockage en mémoire vive. La structure des données stockées permet de l'utiliser comme système de base de données, de cache ou encore de message broker. Les types de données proposés sont évolués. Au-delà de la chaîne de caractères, Redis supporte nativement les hash, les listes, les set, index géospaciaux et autres données structurées. Redis propose nativement un système de publication/souscription permettant d'implémenter simplement des communications asynchrones.
Redis est donc une base de données clef/valeur évoluée. Elle permet également la mise en place de la réplication, prend en charge le langage de script LUA, les transactions, et, plus récemment, un mécanisme de persistance sur disque. Il est possible de mettre en place une



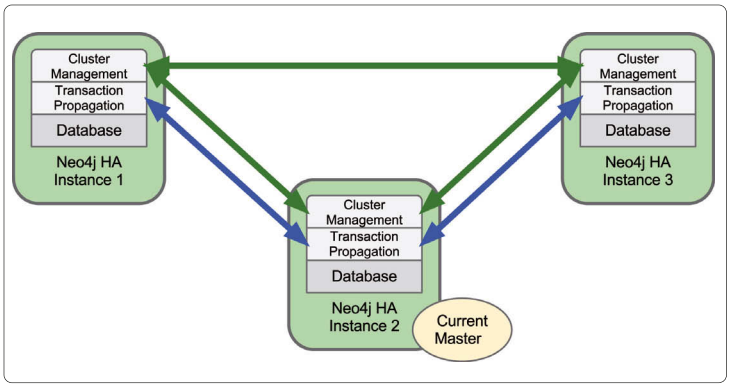
architecture haute disponibilité en s'appuyant sur Redis Sentinel et la répartition automatique avec Redis Cluster.

```
redis> SET key1 "Hello"
OK
redis> MGET key1
1) "Hello"
```

MongoDB : stockage de document

MongoDB est une base de données orientée stockage de documents au format JSON. Cette base propose de nombreux connecteurs pour les différents langages les plus répandus ainsi qu'un Shell permettant de requêter en ligne de commande avec des instructions en langage JavaScript. La grande force de MongoDB est de proposer une API simple permettant de mettre en place facilement des opérations d'agrégations de données.
Du point de vue des performances, MongoDB permet la mise en place d'un cluster appelé « replica set ». Celui-ci permet la duplication des données sur un groupe de serveurs qui possédera un nœud principal et n-serveurs de backup. Si à un moment donné le nœud principal est inopérant, automatiquement l'un des serveurs de backup deviendra serveur primaire.
Les index sont pris en charge dans MongoDB et ont le même rôle et principe de fonctionnement que sur une base SQL. Comme il n'y a pas de schéma, les index sont construits par un appel à l'API MongoDB. Il est également possible de créer des index multichamps. MongoDB prend particulièrement bien en charge les index géospaciaux.
La dernière version de MongoDB, la version 3.2, propose de nombreuses évolutions intéressantes telles que la validation de documents, un moteur de stockage chiffré ou encore des outils tels que des connecteurs pour la Business Intelligence.

```
db.products.find( { qty: { $gt: 25 } } )
```



ElasticSearch : moteur de recherche

La recherche plein-texte fait partie des besoins spécifiques auxquels répondent certains outils. Le plus répandu est ElasticSearch. Ce dernier exploite un moteur Lucene pour l'indexation et la recherche des données. ElasticSearch offre plusieurs avantages dont le principal est de proposer une interface de Web Services. Cette API REST permet ainsi d'échanger des données au format JSON, ce qui permet une interopérabilité dans tous les langages supportant HTTP. ElasticSearch propose également des optimisations de performances au travers la mise en place d'un cache et d'un mécanisme de haute disponibilité. Dans son utilisation, on va retrouver l'équivalent de ce que permet MongoDB. ElasticSearch permet le stockage de documents JSON et leur requêtage. Les opérations de types map/reduce et agrégation sont également disponibles pour un gain de performance significatif. Le revers de la médaille est que cet outil a été pensé pour la recherche : le vocabulaire est donc différent de celui des bases de données et il n'est pas toujours simple de se retrouver dans la documentation. ElasticSearch diffère notamment des bases de données classiques puisqu'il ne propose aucun mécanisme d'authentification ni de règles d'accès. Il faut donc prévoir dans son architecture un développement et une configuration prenant en compte ces contraintes de sécurité.

```
curl -XPUT 'http://localhost:9200/blog/user/dilbert' -d '{ "name" : "Dilbert Brown" }'
```

Neo4j : stockage de graphe

Les graphes font partie des structures les plus complexes qu'il est possible de stocker en base de données. La force de Neo4J est d'offrir un support natif du requêtage de graphes tout en assurant des transactions ACID. Neo4j supporte la mise en place de clusters pour la montée en charge et notamment la gestion du failover pour la disponibilité.

Le principe est donc de matérialiser les relations entre objets lors de l'insertion en base, les performances en lecture sont donc excellentes. Il est facile d'effectuer des requêtes traversant les relations entre les différents noeuds en base. Le stockage des données reste compact et grâce à la mémoire cache, son utilisation peut se faire même sur du matériel aux performances limitées. Les requêtes à la base Neo4J se font par des Web Services HTTP, ce qui rend sont utilisation possible dans tous les langages. Malgré tout, on trouve un client natif pour de nombreux langages : Java, .NET, JavaScript (Node.js), Python, Ruby, PHP et même R, Go, Clojure, Perl, Haskell.

Adressant un besoin spécifique, Neo4J est pris en charge dans de nombreux écosystèmes, dont MongoDB ou ElasticSearch.

TRAITEMENT DE DONNÉES

Souvent associées aux bases de données NoSQL, les solutions de traitement de données sont indissociables des architectures Big Data. En tête de gondole, Hadoop fait bonne figure. Ce framework permet de traiter de larges jeux de données au travers de clusters de serveurs dans un mode distribué. Son objectif est de simplifier la conception d'algorithmes de traitement distribué. Dans l'écosystème Hadoop, on va retrouver un ensemble d'outils spécifiques : Pig (scripting sur les données), Hive (requêtage sur les données), Oozie (ordonnanceur de tâches), Zookeeper (coordination des services), Mahout (apprentissage automatique et datamining). Hadoop étant plutôt conçu pour traiter de larges volumes de données en batch sans objectif de rapidité, le projet Drill permet d'effectuer des requêtes en temps réel. Apache Spark est une alternative plus récente et plus performante qu'Hadoop qui offre une certaine compatibilité.

OrientDB : stockage de graphe et plus encore

OrientDB est un système de base de données sous forme de graphe de seconde génération. Ce qu'il propose c'est tout simplement davantage de flexibilité que Neo4j avec un stockage sous forme de documents. Il propose également tout le nécessaire pour répondre aux problématiques de performances à travers la mise en place d'un cluster avec réplication et sharding.

OrientDB propose également différents niveaux de sécurité et de règles d'accès aux données contrairement à Neo4j. OrientDB est globalement plus performant tout en supportant le langage SQL. Cette base de données permet les jointures et bien d'autres concepts propres à SQL sans pour autant imposer la mise en place d'un schéma. Celui-ci est facultatif et il est même possible de concevoir une base de données avec à la fois un schéma pour certaines données et l'absence de schéma pour d'autres.

Le moteur d'OrientDB supporte les graphes, les documents, les clefs/valeurs et des modèles objets ce qui en fait un outil répondant à de nombreuses problématiques fonctionnelles. Il ne s'agit pas d'un support multi-modèles par couche, mais bel et bien d'un moteur prenant en charge nativement ces différents modèles de données tout en étant capable de travailler sur ces données hétérogènes ensemble. Il ne faut pas s'attendre à un niveau de performance aussi efficace qu'une base de données spécialisée dans un modèle de données.

InfluxDB : données chronologiques

La base de données Influx est une base permettant le stockage de données de type chronologiques. C'est une base idéale pour stocker des données de métrologie (météo, santé, monitoring...) avec un focus particulier sur la haute disponibilité et sur les performances. InfluxDB est une base open source écrite en langage Go, simple à installer. Une interface Web est fournie avec le serveur et permet de réaliser les opérations de base et d'accéder aux données. Les données stockées possèdent un timestamp, un numéro de séquence, un certain nombre de métadonnées sous forme de clef/valeur au format texte et une valeur numérique.

```
INSERT temperature,city=Paris value=11
```

Les requêtes sont simples et proches du langage SQL. Ce langage (InfluxQL) permet d'agréger les données à la volée avec le mot clef GROUP BY :

```
select * from temperature limit 10
SELECT mean(value) FROM temperature WHERE city=Paris GROUP BY time(60m)
```

Le stockage des données peut aussi bien se faire en mémoire vive que sur disque selon les besoins. Influx permet la mise en place d'une politique de rétention spécifique et propose un système de sauvegardes incrémentales.

Riak: haute disponibilité

Riak fait référence à trois produits différents fonctionnant grâce à une API REST. Le plus connu est certainement Riak KV, une base de données de type clef/valeur. Riak KV est une base de données distribuée, scalable de manière linéaire, hautes performances qui visent la meilleure tolérance aux pannes possible. D'un point de vue fonctionnel, elle propose un stockage de données sans schéma. Cette base prend en charge le stockage des données sous différentes formes : texte, images, documents JSON/XML/HTML... Riak est développé en Erlang, un langage répondant au paradigme de programmation concurrente, temps réel et distribué. Le point fort du langage Erlang est justement sa tolérance aux pannes et Riak en profite directement. L'interopérabilité est également très importante et permet à Riak de fonctionner sur de très nombreuses architectures dont Raspberry Pi.
Riak TS permet répond au besoin spécifique de stockage de données chronologiques, ce qui en fait une alternative d'InfluxDB. Celui-ci se différencie d'InfluxDB notamment par l'absence de typage des données qui se limitent à des valeurs numériques et texte. Il est également possible d'exécuter des scripts côté serveurs ainsi que des triggers avant ou après chaque insertion.
Enfin, Riak S2 permet de stocker des objets lourds permettant ainsi aux serveurs Web de s'affranchir d'un système de fichier par nature non scalable pour le stockage d'images ou autres fichiers par exemple. Riak S2 répond ainsi au besoin d'un système de fichier distribué sur le Web, avec notamment une API compatible avec Amazon S3.

Cassandra : stockage orienté colonnes

Apache Cassandra est une base de données permettant de stocker différents types de données structurées ou non. Elle permet la répartition à travers plusieurs serveurs avec de bonnes capacités montées en charge et haute disponibilité. Les opérations de maintenance sont simples à mettre en oeuvre notamment à chaud. Cassandra ne contient pas de SPof (Single Point of Failure). Le modèle de donnée flexible apporte une souplesse dans la conception des données tout en offrant de bons temps de réponse. Initialement développé par Facebook en Java, Cassandra est distribué sous licence Open source et fait partie des solutions NoSQL les plus utilisées. Les données sont stockées sous forme de clef/valeur en s'appuyant sur une architecture relationnelle orientée colonne. Les données sont distribuées selon leur clé primaire. Le langage de requête est proche du SQL et porte le nom de CQL (Cassandra Query Language).

```
CREATE COLUMNFAMILY MesColonnes (id text, Nom text, Prenom text, PRIMARY KEY(id));

INSERT INTO MesColonnes (id, Nom, Prenom) VALUES ('1', 'Doe', 'John');

SELECT * FROM MesColonnes;
```

ArangoDB : multimodèles

ArangoDB est une base de données permettant le stockage de données multi-modèles. Son moteur est suffisamment flexible pour permettre le stockage de données hétérogènes : documents, graphes ou encore clef/valeurs. Malgré tout, son approche en fait davantage une base orientée documents, tant en termes de vocabulaire que d'architecture. Pour autant, ArangoDB se différencie par son Langage AQL qui ressemble beaucoup au langage SQL tout en s'appuyant sur le format JSON :

```
INSERT { name: "John Doe", gender: "m" } IN users
```

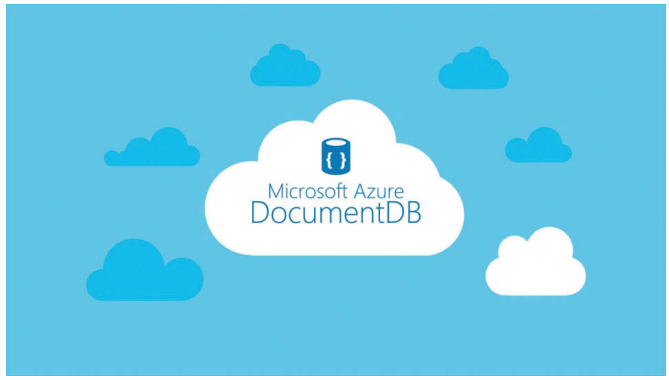
L'agrégation est implémentée dans une approche différente de MongoDB, en conservant une syntaxe proche du SQL grâce à des mots clefs spécifiques comme FILTER et COLLECT :

```
FOR u IN users
  FILTER u.age > 30
  FOR c IN cities
    FILTER u.cityId = c.id
  RETURN { user: u, city: city }

  FOR u IN users
    FILTER u.active == true
    COLLECT age = u.age INTO usersByAge
    SORT age DESC LIMIT 0, 5
    RETURN {
      "age" : age,
      "users" : usersByAge[*].u.name
    }
```

Essentiellement documenté pour JavaScript et notamment pour leur propre framework Foxx. ArangoDB est documenté pour Node.js et PHP et ne propose malheureusement pas de nombreux connecteurs pour le moment.

MariaDB : SQL n'est pas mort

MariaDB est tout simplement un serveur de base de données dont le fonctionnement permet de remplacer MySQL. Il apporte des fonctionnalités supplémentaires tout en conservant la meilleure compatibilité. MariaDB propose notamment de nombreux moteurs, dont XtraDB qui remplace avantageusement InnoDB. SphinxSE permet une recherche textuelle performante. Le stockage Big Data est également mis à disposition avec le support du moteur TokuDB. Le moteur Spider quant à lui propose un support natif du sharding. Globalement des efforts sont faits pour intégrer des moteurs NoSQL avec notamment le support de Cassandra.



	Format	Transactions	Relationnel	Haute disponibilité	Fonctions avancées
Redis	Clef/valeur	Non	Non	cluster	Communication asynchrone
MongoDB	Document JSON	Non	Non	cluster	Agrégation
ElasticSearch	Document JSON	Non	Non	cluster	Agrégation
Neo4j	Graphe	Oui	Oui	maître/esclave	intégration MongoDB/Spark/ElasticSearch
OrientDB	Multi-format	Oui	Oui	cluster	transactions et sécurité
ArangoDB	Multi-format	Oui	Oui	cluster	jointures
InfluxDB	Série temporelle	Non	Non	cluster (expérimental)	outils de visualisation fournis
Riak	Série temporelle, clef-valeur	Non	Non	cluster	tolérance aux pannes
Cassandra	Colonnes	Non	Oui	cluster	réplication avancée
MariaDB	SQL	Oui	Oui	maître/esclave	Multi-moteurs
PostgreSQL	SQL	Oui	Oui	maître/esclave	support JSON, géolocalisation

Les développeurs de MariaDB ont travaillé sur l'optimisation de la base de données ce qui permet d'atteindre un meilleur niveau de performance que MySQL. Cela s'explique essentiellement par l'optimisation des moteurs de stockage, mais un travail a également été fourni sur la gestion du pool de connexions et une meilleure prise en charge des requêtes exécutées en parallèle sur un cluster de serveurs répliqués. MariaDB offre également des fonctionnalités supplémentaires, que ce soit au niveau de la gestion des droits utilisateurs, de la création et gestion des schémas ou encore sur le requêtage. On peut notamment citer la prise en charge de la géolocalisation. MariaDB se veut également plus stable et plus sûr que MySQL.
MariaDB est disponible sur la plateforme Windows Azure via le service MariaDB Enterprise Cluster.

SQL : zoom PostgreSQL

PostgreSQL qui présentait déjà beaucoup d'intérêt par rapport à MySQL fait partie des bases de données SQL qui ont su évoluer et proposer des fonctionnalités pertinentes qui font face aux meilleures de NoSQL. Il gère notamment particulièrement bien les données temporelles et permet le stockage d'attribut au format JSON pouvant être requêté en SQL. Il y a peu de bases de données qui se comportent aussi bien avec la géolocalisation. Postgresql est optimisé pour traiter ces données et c'est d'ailleurs pour cette raison qu'il est utilisé par OpenStreetmap.
L'utilisation de Postgresql en tant que base de documents JSON est possible. On peut si nécessaire envoyer des requêtes classiques en SQL et récupérer du JSON. Cependant il ne faut pas comparer hâtivement Postgresql à Mongodb; certaines fonctionnalités ne sont pas supportées et ne le seront probablement jamais. À commencer par l'atomicité de mise à jour des attributs JSON. Avec Postgresql on ne peut pas modifier partiellement un document JSON, on ne peut que le remplacer intégralement. Mongodb permet d'effectuer des insertions concurrentes ce que ne permet pas Postgresql puisqu'il respecte les propriétés ACID : les événements ont lieu les uns après les autres.
D'un point de vue des performances, il est possible de mettre en place un cluster avec des noeuds maîtres/esclaves, mais la haute disponibilité reste limitée dans ces conditions.
PostgreSQL supporte les tables partitionnées, cela permet une ségrégation des données selon un critère de date, de valeur numérique ou de critère personnalisé. Il s'agit de vues personnalisées qui seront plus rapides en lecture et qui héritent d'une table existante :

```
CREATE TABLE "Weather_Z0" (
CHECK (  left("Zip", 1) = '0' )
) INHERITS ("Weather");
```

Les services Amazon AWS

Amazon propose différents services d'hébergement de bases de données propriétaires. Parmi ceux-ci Aurora, un moteur de stockage compatible MySQL 5.6. C'est Amazon RDS qui gère les tâches liées à la gestion des bases de données Aurora: mise en service, sauvegardes et récupérations, détection de pannes et réparation. L'intérêt ici est de permettre aux utilisateurs de migrer leur application sans avoir à modifier le code source. L'interface d'administration permet de facilement gérer les opérations courantes ainsi que la montée en charge.
Amazon CloudSearch est un moteur de recherche qui permet de créer un domaine (équivalent base de données) et d'y associer une source de données : fichiers, S3, DynamoDB... L'outil d'initialisation permet alors de définir les champs de recherche sur les données et d'y associer un type (numérique, texte, date...) et une langue éventuelle parmi la trentaine proposée. La recherche s'effectue parmi l'ensemble des champs qu'il est possible de filtrer. L'API propriétaire peut être utilisée pour des tests à travers la console, mais un SDK est fourni ainsi qu'une interface de Web Services REST au format JSON/XML. CloudSearch propose des fonctionnalités similaires à ElasticSearch à la différence principale qu'il ne permet pas le stockage de documents riches et ne propose pas de suggestions en cas de fautes de frappe ou de termes approchants.
Amazon Redshift est un service de recherche Big Data. Il permet le stockage et l'analyse de grands volumes de données de plusieurs Téraoctets. Le langage de requêtage retenu est, là encore, le SQL, associé à une procédure spécifique pour le chargement des données. Son utilisation reste simple puisqu'il peut être utilisé avec les outils habituels et les connecteurs ODBC/JDBC standards. Amazon met à disposition l'intégration de différentes solutions d'informatique décisionnelle du marché pour aller plus loin et construire de véritables applications BI : Talend, Informatica, Pentaho, QlikView...
SimpleDB est la solution de base de donnée NoSQL d'Amazon. Elle permet le stockage de données non relationnelles. Pensée pour la haute disponibilité, en arrière-plan, Amazon s'occupe de la réplication et d'une répartition géographique des données. Techniquement, SimpleDB stocke des données sous forme de table de ligne clef/valeur. Il permet la réalisation de requêtes simples d'insertion, de suppression et de sélection. Étant donné que les données sont non relationnelles, il n'est pas possible d'effectuer des requêtes complexes s'apparentant à des jointures ou à de l'agrégation.
Un autre service AWS et non des moindres: DynamoDB. Amazon nous propose un kit de développement logiciel documenté qui nous permet d'intégrer son utilisation au sein d'une application Java, .NET ou PHP. Un Shell est également disponible. DynamoDB est une solution simple à

mettre en oeuvre qui inclue un hébergement scalable. Les fonctionnalités proposées restent par contre limitées en comparaison de ses concurrents open source comme MongoDB ou ElasticSearch.

Les services Google

Google BigQuery est une solution en ligne qui permet le requêtage massif de jeux de données de grands volumes. Le principe de BigTable est de simplifier le fonctionnement SQL en ne proposant pas le stockage dans la table accessible en écriture, mais en ajout de lignes seulement. On peut accéder à BigQuery par une interface graphique, ou bien par un outil en ligne de commande, ou au sein d'une application grâce à l'API HTTP. Différents outils permettent de visualiser les données ou encore de simplifier le chargement de données. Les données peuvent être chargées ou exportées au format JSON et CSV. BigQuery est un outil très utile pour traiter de la donnée, mais n'est pas utilisable en lieu et place d'une base MySQL pour une application Web, par exemple un déploiement Wordpress. Google Cloud Bigtable est le service d'hébergement de données NoSQL de Google. Il s'agit d'un système qui est utilisé par la firme pour ses propres applications incluant son moteur de recherche, Google Analytics, Google Maps et GMail. Autant dire que ce service a été conçu dans un seul but : assurer une disponibilité maximale avec des temps de réponse courts malgré une forte charge. Ceci en fait une solution de choix pour le développement d'applications qui traitent des données en temps réel. Techniquement, BigTable s'intègre avec Hadoop et Spark aussi bien qu'avec les autres services qu'ils proposent, dont BigQuery. Concrètement, BigTable supporte l'API standard de la base de données HBase. Il s'agit d'une base de données orientée colonne tout comme Cassandra.

Bigtable est adapté pour les grands volumes de données, mais ne conviendra pas pour une utilisation classique de base de donnée. Google propose une autre solution NoSQL baptisée Google Cloud Datastore. Il s'agit d'une base de données prenant en charge le sharding et la réplication automatique pour mieux appréhender les montées en charge. L'API proposée s'approche du SQL, et, surtout, ce système permet des transactions ACID ainsi qu'une gestion classique des index.

```
var companies = query.filter('name =', 'Google').filter('size <', 400);
```

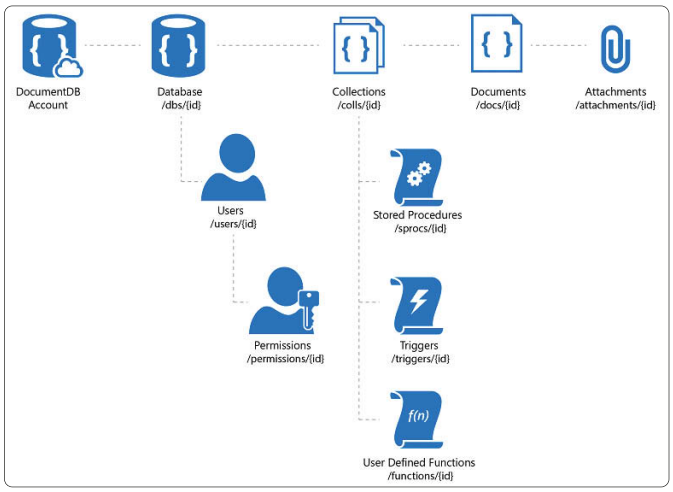
Les services Microsoft

DocumentDB est la solution NoSQL de Microsoft. Ce service est uniquement disponible en ligne et propose de stocker des documents JSON et de les requêter de façon simple et immédiate. Le schéma n'est pas fixé, mais la grammaire du langage de requêtage est proche du langage SQL. Il est également possible d'écrire des procédures stockées, triggers et autres fonctions en JavaScript. Finalement le vocabulaire et le fonctionnement de DocumentDB s'approchent de MongoDB. Cependant il existe des différences majeures comme l'absence de Shell ou encore des limitations concernant les capacités de DocumentDB à effectuer des opérations de Map/Reduce ou d'agrégation. Microsoft Azure Search est un service de recherche hébergé au sein de la plateforme Azure. Il permet une implémentation simplifiée d'un moteur de recherche au sein d'une application Web ou mobile tout comme le propose Amazon Cloud Search. Il s'agit donc là aussi d'une solution clef en main qui répondrait à la mise en place d'ElasticSearch. Le service est accessible au travers d'une API HTTP ou du SDK .NET. Microsoft se démarque ici en proposant 56 langues contre seulement 34 pour Amazon et propose les suggestions de recherche ainsi que l'autocomplétion. Un index géospatial permet également de filtrer et de trier les données selon la zone géographique. Microsoft tire très certainement cette avance de son expérience dans le développement dans son moteur de recherche Bing.

Microsoft tout comme ses concurrents, propose une solution SQL en ligne s'appuyant sur sa solution SQL Server. Cette solution s'intègre dans sa plateforme Azure et porte le nom Azure SQL Database. L'hébergement d'une application fonctionnant avec SQL Server sera transparent et permettra de profiter des fonctionnalités d'administration intégrées. Au-delà du système de tarification progressif suivant la consommation, Microsoft met à notre disposition un système appelé Elastic Pool permettant une configuration flexible des ressources.

NOUVEAUTÉS SQL SERVER 2016

La nouvelle mouture de SQL Server apporte son lot d'innovations. Tout d'abord, c'est l'arrivée des bases de données en mémoire vive qui permettra d'améliorer les performances des applications statistiques temps réel. Un nouveau système de chiffrement voit également le jour et permet de protéger les données stockées sur les serveurs. La nouvelle technologie Stretch Database permet de migrer de manière dynamique vos données à chaud ou à froid vers Microsoft Azure. Une des grandes nouveautés de SQL Server 2016 est la mise à disposition de Polybase, un nouveau service qui permet d'utiliser des données disponibles dans SQL Server ou un cluster Hadoop dans un Windows Azure. Des améliorations de performances ont été apportées sur les sauvegardes ainsi que sur la disponibilité et sur la reprise après sinistre.



Conclusion

Comme nous l'avons vu, SQL a encore de beaux jours devant lui et les technologies NoSQL trouvent leur place sur des besoins spécifiques qu'ils adressent. Nous avons pu parcourir les principales solutions du marché, mais il en existe beaucoup d'autres. Pour aller plus loin, vous pouvez aussi regarder les solutions Couchbase, HBase, Accumulo, Voldemort, mais également les solutions de cache Memcached, Hazelcast. Qu'il s'agisse de serveurs dédiés sur une infrastructure publique ou privée ou encore de solutions cloud qui vous proposent un service de stockage et de requêtage clef en main. Ne succombez pas trop hâtivement aux sirènes du NoSQL. Côté hébergement en ligne, nous n'avons pas fait le tour de toutes les solutions. Oracle propose également une plateforme en ligne avec notamment Oracle Coherence sur sa plateforme Java Cloud Service et il existe des acteurs de taille plus restreinte comme Heroku ou le français Clever Cloud. N'hésitez pas à essayer et à comparer ! ⊞

ChakraCore s'ouvre !

Depuis l'arrivée de Satya Nadalla aux commandes de Microsoft, en février 2014, on peut dire sans peine que de grands changements se font ressentir. Le successeur de Steve Ballmer a pris le pari de révolutionner l'éditeur. Une révolution qui n'est pas que superficielle ou basée sur du marketing bien emballé. Nous sommes dans l'ère du MS v2.0 avec un remaniement important impactant jusqu'à l'ADN de l'entreprise.

Freddy Hebrard
mcpd .net, Chef de projet, formateur IT

Jusqu'à présent notre CEO a su créer l'engouement des consommateurs sur des produits « haut de gamme » comme la catégorie PRO des Surfaces. Il a su susciter notre intérêt mais aussi celui d'un public plus large en axant Microsoft sur le marché des smartphones premiums avec les nouveaux Lumia.

On se souvient de la phrase-choc : « Microsoft aime Linux », ce qui n'était encore hier que raillerie, insulte, voire un blasphème pour l'éditeur, est devenu aujourd'hui un véritable cheval de bataille : l'open source, une arme nécessaire à un profond changement afin de rester compétitif face à d'autres leaders du marchés.

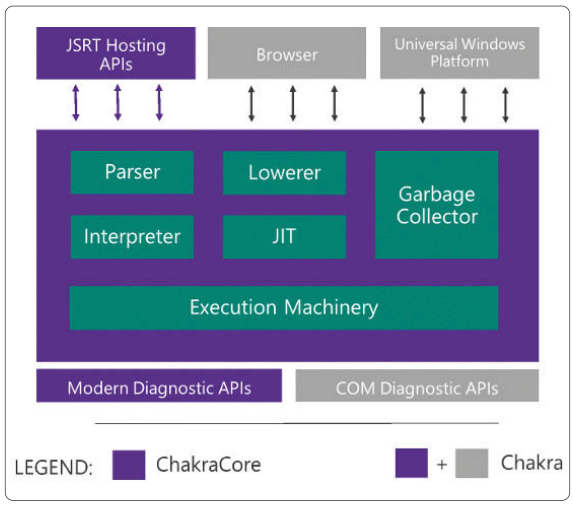
Nous avons eu droit à MS Build, Visual Studio Code, une partie importante du framework .Net (.Net Core). Enfin, en décembre dernier, lors de la conférence JSConf, c'était au tour de Chakra d'être mis en open source, sous licence MIT. Il ne s'agit pas de tout Chakra mais d'une petite partie du moteur JavaScript, utilisé dans Edge (Windows 10), projet sobrement appelé ChakraCore.

Quand Microsoft ouvre son chakra

Chakra vous le connaissez tous, c'est le moteur JavaScript utilisé par Edge, mais aussi au sein de Windows 10 pour la «Universal Windows platform ». Microsoft propose donc d'ouvrir quelques modules et ce projet se nomme ChakraCore. Il est accessible dès aujourd'hui sur GitHub : github.com/Microsoft/ChakraCore. Pour l'instant il est accessible uniquement pour les plateformes Windows à partir de Windows 7 sp1. Il y a une réelle ambition de l'ouvrir sur d'autres plateformes comme Linux (notamment sur Ubuntu).

Mais qu'est-ce que chakracore ?

C'est une machine virtuelle autonome pouvant faire fonctionner diverses applications, comme des jeux, des services (par exemple dans un Cloud), mais aussi au sein de objets connectés : exemple concret dans Windows IoT, Nodejs tourne sous le moteur Chakra. Actuellement Nodejs fonctionne avec un moteur venant de Google (V8). Selon les mots de Microsoft : il a été désigné pour être intégré dans n'importe quelle application nécessitant un moteur rapide, léger, et évolutif.



ChakraCore, le contenu

En prenant cette machine virtuelle, vous aurez accès à :

- Un compilateur JIT (Just in time) pour plateforme x86, X64 et ARM ;
- Du garbage collector (ramasse-miettes) ;
- Prise en charge des API Javascript Runtime (JSRT) qui permettent l'intégration rapide dans vos applications ;
- Large éventail de fonctionnalités JavaScript ;
- Un parseur JavaScript .

À noter que cette version Core n'est pas livrée avec les liens privés vers le navigateur et encore moins le système utilisé par l'Universal Windows Platform.

Une implication importante des développeurs et de la communauté

« L'histoire dont vous êtes le héros », ça ne vous rappelle rien ? Voilà ce que l'on vous propose amis développeur : vous avez des idées, vous voulez voir certaines fonctionnalités sur cette plateforme, faites une proposition ! Si elle est retenue, elle sera aussi portée sur la version fermée de Chakra, donc susceptible d'être utilisée dans Edge ou Universal Windows Platform.

Pourquoi garder un œil attentif sur cette technologie ?

Clairement ce projet a le vent en poupe, nous savons que Microsoft fournit un gros travail pour intégrer les dernières fonctionnalités de JavaScript et tend toujours vers cette amélioration afin de coller le plus rapidement et le plus fidèlement au support avancé de l'ECMAScript 2015 (ES6).

Aujourd'hui des groupes comme Intel, AMD ou nodeSource ont clairement indiqué leur intérêt pour une contribution dans ce projet.

Les performances

Vous devez certainement vous rappeler les propos tenus lors de la sortie imminente d'Edge à la conférence Build 2015 qui voulaient démontrer une performance impressionnante face aux concurrents et plus particulièrement à V8 de Google. Afin d'être un peu plus dans la démonstration, on peut voir sur d'excellents sites des comparaisons via octane 2 et Jet Stream démontrant la supériorité de Chakra à l'heure actuelle.

Cette transition vers l'open source pour ce moteur JavaScript est une stratégie importante, les frameworks comme Nodejs ont le vent en poupe et Microsoft l'a bien compris. Tout cela vous laisse imaginer le potentiel qu'il y a dessous et l'intérêt que nous avons en tant que développeurs de garder un peu de notre temps pour garder un œil dessus.

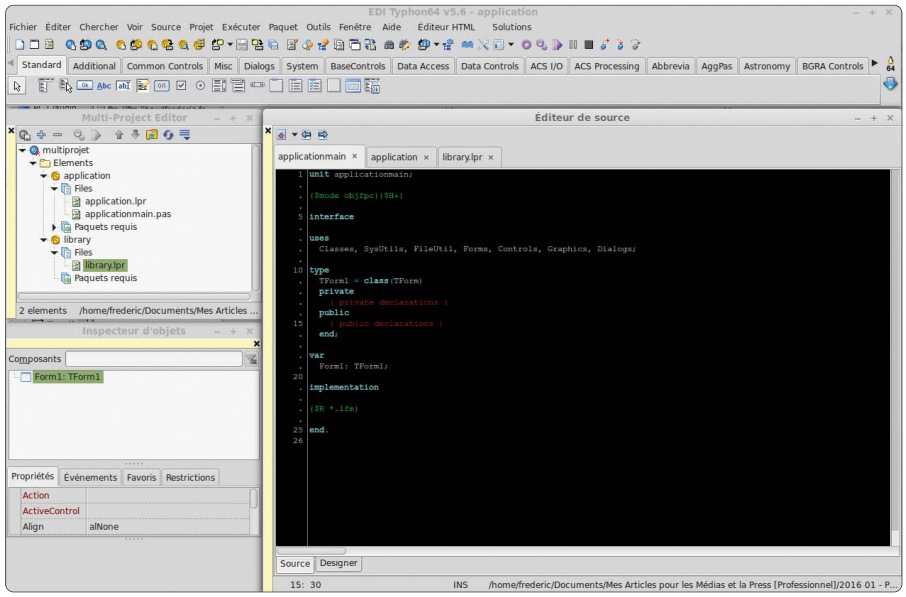
Et après ?

Microsoft compte vraiment sur ce projet, afin de fédérer la communauté des développeurs en les impliquant dans le processus d'évolution du produit. Et d'après la roadmap accessible sur GITHUB (https://github.com/Microsoft/ChakraCore/wiki/Roadmap), de grands chantiers sont à venir pour juin 2016, un timing serré qui démontre l'investissement dans ce projet.

Delphi, l'itinéraire d'un RAD

En créant Delphi, Borland fut bien inspiré mais, ne partait pas de rien. L'éditeur disposait déjà d'une large base d'utilisateurs avec son Turbo Pascal.

Frédéric Libaud
Expert indépendant en numérique, avec plus de 20 ans de métier de l'ingénierie logicielle sur des environnements comme Delphi ou C++ Builder et en infrastructure. Pour en savoir plus : http://about.me/frederiClibaud.



Le Pascal Objet est en effet un langage solide, proche du C, inventé dans les années 1970 pour l'enseignement. Et c'est Borland avec le Turbo Pascal, qui lui donnera ses lettres de noblesse. D'ailleurs diverses versions furent créées pour différentes plateformes; même Microsoft proposa fut un temps un compilateur Pascal mais, avec un succès bien moindre.

VBK

Delphi dont le nom de code pendant son développement était VBK, ce qui signifie Visual Basic Killer, a eu un large succès. Lancé en 1995, il proposait une approche orientée RAD (développement rapide d'application) sous Windows, à l'instar de Visual Basic dont il se voulait le concurrent direct.

De nombreux projets ont été développés avec Delphi, par des éditeurs ou de grandes entreprises.

Non, le Pascal Objet n'est pas mort

En 1993, parallèlement Florian Klämpfl lance un projet de compilateur Pascal Objet nommé FPK Pascal, qui deviendra plus tard, en 1996, Free Pascal. Largement compatible avec le Turbo Pascal et Delphi, il permet un développement multiplateforme/multi-architecture (ARM, Intel, PowerPC, ...). FPC est distribué sous licence Open Source GNU.

Free Pascal a évolué au fil du temps, sa dernière version apporte de nombreuses fonctionnalités. Il est ainsi possible tout comme pour Delphi de compiler avec différents niveaux d'optimisation. Au niveau débogage, beaucoup d'options sont possibles, même l'intégration d'informations pour GpProfile ou Valgrind.

Delphi est sur Windows et ailleurs ?

Au début des années 2000, Borland fait une tentative de commercialisation d'outil RAD équivalent à Delphi pour Linux, il s'agit de Kylix. Ce dernier n'eut malheureusement pas le même succès que son grand frère.

En parallèle, Borland commercialise C++ Builder et C++ BuilderX, s'appuyant respectivement sur le C++ et Java mais, exploitant la même philosophie. C++ Builder avait pour particularité d'avoir la capacité de compiler à la fois du code C++ et Pascal.

Puis Borland regroupa ses produits de développements dans une nouvelle filiale, CodeGear, qui fut ensuite vendue à Embarcadero. Cette dernière ayant été rachetée en fin d'année dernière par IDERA.

Delphi en est à sa version 10, nommée Seattle. Les précédentes versions de Delphi à partir de la XE2 intègrent un nouveau framework appelé Firemonkey, qui permet le développement d'applications multiplateformes. Principalement pour Windows, OSX, iOS. A noter qu'il faut avoir l'équipement cible pour pouvoir faire la génération. On ne peut donc utiliser Delphi sur d'autres systèmes, sauf à utiliser des sessions virtualisées de Windows.

Les alternatives à Delphi

Cela inspira des développeurs qui voulurent un outil équivalant à Delphi, c'est ainsi qu'est né Lazarus, également sous licence Open Source LGPL.

Cet outil s'appuie à l'instar de Delphi sur une bibliothèque de composants graphiques et non graphiques pour fonctionner, la LCL (Lazarus Component Library). Lazarus est Multiplateforme, on peut l'utiliser sur Linux (Arch, CentOS, debian, Mint, OpenSuSE, Ubuntu, ...), OSX et Windows, puisqu'il s'appuie sur FPC (Free Pascal Compiler). Il est d'ailleurs généré avec ce dernier. La LCL permet donc la conception d'applications pour les différents systèmes supportés par FPC.

A l'instar de Delphi, Lazarus permet le développement d'applications diverses (console, graphique, serveur, ...) en mode RAD avec une compilation rapide.

Lazarus permet également le développement d'applications pour Android et iOS.

Et les autres alors ?

D'autres développeurs se lancèrent et créèrent MSEide ou CodeTyphon Studio.

Ce dernier s'appuie sur le binôme Free Pascal et Lazarus; il est distribué sous licence Freeware.

CodeTyphon est un environnement de haut niveau à l'instar de Delphi et Lazarus qui apporte en plus la possibilité de faire de la Cross Compilation pour de multiples systèmes cibles supportés par FPC, mais aussi Java.

Le développeur peut ainsi créer une application supportant différentes cibles (couple plateforme/architecture) et générer depuis la plateforme de son choix.

CodeTyphon Studio peut sembler un peu rébarbatif à installer, car il faut télécharger une archive, la décompresser. Il faut ensuite procéder à l'installation de l'ensemble, en plusieurs étapes,

avec des droits d'administrateur.
La dernière version de l'environnement de CodeTyphon V-IDE, dorénavant en version 5.60, s'appuie sur la dernière version du compilateur Free Pascal (3.0.0) et de Lazarus (1.4.4). Elle supporte naturellement Windows en 32 et 64 bits mais, aussi QT (3, 4 et 5) et GTK (2, 3 et 4).
L'interface de l'environnement de développement est assez proche de celle de Lazarus mais, apporte plus de sophistication. Une des grandes nouveautés de la dernière version, c'est la gestion de projets multiples, une application et une librairie, par exemple. A l'instar de Microsoft Visual Studio, ce que ne propose pas Lazarus pour l'instant.
Comme je l'écris plus haut, Typhon V-IDE permet la cross-compilation d'application sur le système de développement. Ce qui permet au développeur de créer des applications pour différentes cibles, de les coder et de les compiler sur la même machine, sans disposer forcément des équipements.
Naturellement qui dit compilation multi-cibles, dit compilations conditionnelles, tout comme dans Lazarus d'ailleurs, grâce au support de FPC.
CodeTyphon est donc une alternative sérieuse à Delphi, compte tenu de la gratuité de sa licence ; espérons que cela dure.

L'avenir...

Free Pascal devrait à l'avenir permettre la compilation de code C++, ou tout du moins faire l'édition des liens de modules objets générés dans ce langage. Alors qu'il est déjà possible de le faire avec du code assembleur, tout comme pour Delphi.
Ce dernier n'a pas un avenir encore clairement défini suite au rachat d'Embarcadero par IDERA. On est toutefois en droit de penser, compte tenu de la volonté affichée de cette dernière, que l'environnement va évoluer de façon positive afin de coller au mieux aux attentes de ses utilisateurs. Ce qui en fera à nouveau, un outil majeur sur le marché des RAD. ⊞

Références :

CodeTyphon : http://www.pilotlogic.com
Delphi : http://www.embarcadero.com/fr
Free Pascal : http://www.freepascal.org/
Lazarus : http://www.lazarus.freepascal.org
http://sourceforge.net/projects/lazarus
MSEide : http://sourceforge.net/projects/mseide-msegui/ / https://gitlab.com/mseide-msegui/mseide-msegui

		Delphi	Lazarus	CodeTyphon Studio
Editeur		Embarcadero	Lazarus	PilotLogic
Version		10 Seattle	1.4.4	5.6
Compilateur		Delphi	FPC	FPC
CPU cible	ARM	X	X	X
	i386	X	X	X
	PowerPC		X	X
	PowerPC64		X	X
	SPARC		X	X
	AMD64/Intel 64/X86_64		X	X
	i8086		X	X
	MIPS		X	X
Système d'exploitation cible	Android	X	X	X
	iOS	X		
	Linux		X	X
	Nintendo WII		X	X
	OSX	X	X	X
	Solaris		X	X
	WinCE		X	X
	Windows	X	X	X
Compatibilité RTL		X	X	X
Frameworks	VCL	X		
	Firemonkey	X		
	LCL		X	X
	Java			X
	GTK		X	X
	QT		X	X
	WinRT	X	X	X
Système d'exploitation supporté		Windows 7 SP1, Windows Server 2008	Linux, OSX, Windows	Linux, OSX, Windows
Librairie de composants		VCL, FMX	LCL	LCL
Base de données		FireDAC	Multiples	Multiples

A la découverte de CodeTyphon Studio

J'ai découvert par hasard CodeTyphon Studio en 2015 au fil de mes recherches. Ayant développé sur Delphi pendant plus de 10 ans, la curiosité m'a poussé à regarder ce que l'outil avait dans le ventre. Et bien grande fut ma surprise, sachant qu'à l'époque je n'étais pas pleinement satisfait de ce que j'utilisais et que je cherchais donc à changer de plateforme.

Frédéric Libaud
Expert indépendant en numérique, avec plus de 20 ans de métier de l'ingénierie logicielle sur des environnements comme Delphi ou C++ Builder et en infrastructure. Pour en savoir plus : http://about.me/frederiklibaud.

En effet, en dehors de quelques petites imperfections mais, on en excusera l'éditeur qui n'a pas la taille d'un IBM ou Microsoft, l'outil permet de faire du développement au niveau professionnel.

CodeTyphon Studio un environnement RAD

CodeTyphon Studio est un environnement complet de développement d'applications, un EDI orienté RAD. Il fait partie de ces platesformes comme Microsoft Visual Studio ou Embarcadero Delphi. S'appuyant sur le compilateur Free Pascal et l'environnement Lazarus, tous deux en Open Source, il permet la création d'applications codées en Pascal Objet.

Il fonctionne sous Linux, OS X et Windows. Il permet également de générer des applications fonctionnant à la fois dans ces environnements mais, également sur des cibles comme Android, iOS, Solaris ou la console Nintendo WII.

L'installation

L'installation est la phase la plus rébarbative pour pouvoir utiliser CodeTyphon. Fort heureusement on ne le fait pas souvent et il est possible de l'automatiser avec un script.

Après avoir téléchargé l'archive de la dernière version, l'avoir décompressée, il faudra lancer le script Install.bat sous Windows, Install.sh sous Linux et OS X, ce avec des droits d'administrateur. Ce script procédera à l'installation de CodeTyphon Center, outil qui permettra ensuite le téléchargement, l'installation et le paramétrage de l'ensemble des outils que propose la plateforme. Tout particulièrement de l'environnement RAD, CodeTyphon V-IDE, soit version « SmallIDE » soit « BigIDE ».

CodeTyphon Center permet également le téléchargement et la génération des « cross toolchains », c'est-à-dire des outils qui permettront la génération pour d'autres cibles que le système en cours sur lequel on installe l'environnement.

CodeTyphon Center dispose d'un outil de gestion des mises à jour, permettant ainsi de les appliquer plus aisément.

L'EDI RAD

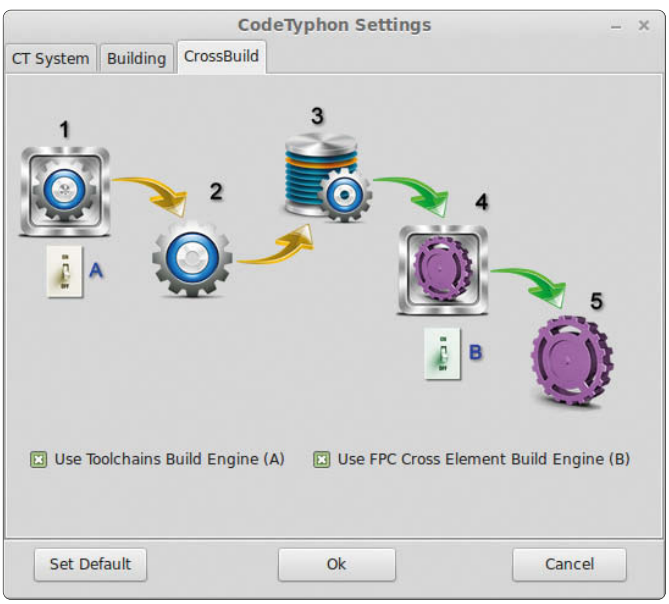
CodeTyphon V-IDE dispose d'un environnement ergonomiquement très proche de Lazarus et donc de Delphi. Les développeurs habitués à ce dernier ne seront pas perdus, ce qui permet une transition en douceur.

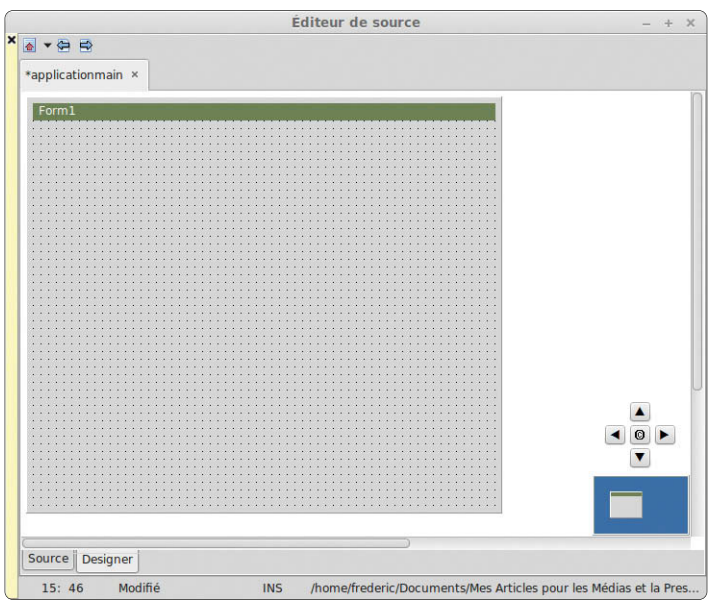
On y retrouve les composants graphiques standards de la LCL, ainsi que de nombreux autres qui seront installés en fonction de la version générée. En effet, pour disposer de toute la palette de composants il faudra générer une version « BigIDE » de l'environnement.

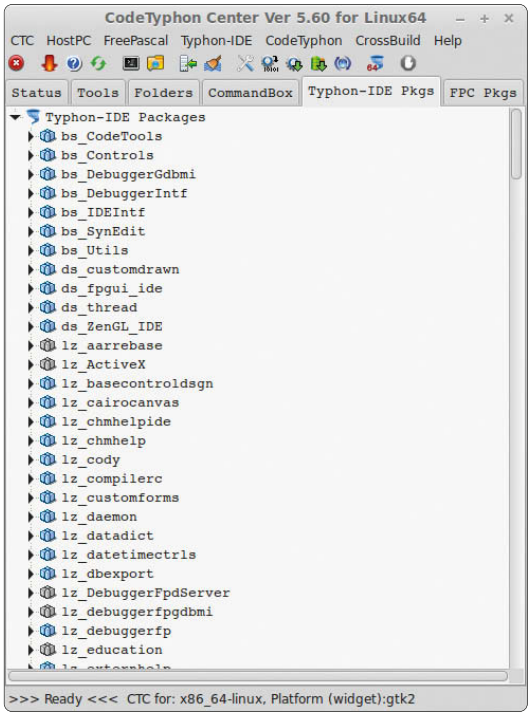
La LCL est un framework de composants portables qui implémente la même architecture de base que la VCL. Ce qui facilite grandement la vie du développeur et permet la migration d'application Delphi en douceur ; l'environnement dispose d'ailleurs d'un assistant de migration de projet. Une application utilisant les composants de base de Delphi, pourra ainsi être migrée et portée sur d'autres écosystèmes sans trop d'efforts. CodeTyphon embarque également de nombreux outils complémentaires comme H2Pas pour la conversion des fichiers d'entête en C/C++. Ressources Builder pour l'édition des fichiers de ressources, un éditeur hexadécimal, un comparateur de fichier, etc.

La conception

Le développeur travaillera avec CodeTyphon V-IDE de la même manière que Delphi ou Lazarus. Les éditeurs de codes et de conception graphique fonctionnent exactement de la même manière, par glisser/déposer des composants pour ce dernier. Les options de l'éditeur sont très nombreuses et paramétrables, à tel point que je n'en ai pas encore fait le tour.







L'environnement permet la création d'applications et librairies bien sûr, mais aussi d'applications CGI et FastCGI, de serveurs HTTP, de modules Web HTML, de modules Apache ou de services (daemon).

Le développeur a à sa disposition tous les outils nécessaires (complétion du code, éditeur de « Todo », outils de recherche, commenter/décommenter un bloc, renommage, ...).

La génération

On génère l'application pour la cible en cours d'utilisation ou pour toutes les cibles. Il y a beaucoup d'options possibles en termes d'optimisation et de débogage. Les chemins des cibles, nom du binaire, sont paramétrables tout comme le chemin de recherche des unités.

On pourra ainsi générer les versions de débogage, commerciales et autres d'un produit complet en une seule passe. Il est aussi possible d'exécuter un script avant et après l'exécution du compilateur.

CodeTyphon Studio exploite au mieux les possibilités du compilateur Free Pascal, de la LCL et des composants embarqués. Tout comme dans Delphi, on embarque les composants graphiques dans un paquet, qui pourra être de conception, d'exécution ou les deux. D'ailleurs un gestionnaire graphique de paquets est disponible. Ce qui permet de concevoir des projets modulaires, quelle que soit l'architecture cible.

Tests et débogage

Le développeur ayant pratiqué Delphi, Turbo Pascal ou Lazarus ne sera pas dépaysé pour tester son projet. Tout y est, pile d'appel, point de suivi, outil d'évaluation, point d'arrêt, fenêtre de code assembleur, etc. Les points d'arrêt sont paramétrables, peuvent être conditionnels, activables/désactivables, etc. Sur un point d'arrêt, on pourra également faire l'enregistrement de la pile d'appel. Le développeur pourra progresser dans le code en pas à pas, pas à pas approfondi, jusqu'au curseur, etc.

Conclusion

On connaît la solidité des applications créées avec Delphi, depuis la première version. CodeTyphon ne déroge pas à cette qualité, en s'appuyant sur le compilateur Free Pascal, Lazarus et la LCL. C'est un outil de choix pour le développeur qui souhaite passer sur une solution fonctionnant sur un autre environnement que Windows où tout du moins permettant la génération de projets multicibles.

Références :

CodeTyphon : http://www.pilotlogic.com
Free Pascal : http://www.freepascal.org/
Lazarus : http://www.lazarus.freepascal.org/
http://sourceforge.net/projects/lazarus

EFREE PASCAL COMPILER

Free Pascal Compiler (FPC) est un projet de compilateur multiplateforme, implémentant le Pascal Objet initié par Florian Kl'ampfl, sous licence Open Source.

Architectures et cibles

Disponible en version 32 et 64 bits, il supporte les architectures X86, PowerPC, ARM, Sparc. D'autres architectures sont prévues mais pas encore supportées. Les principaux systèmes cibles supportés son Linux, iOS, OS X et Windows.

Le compilateur supporte de nombreuses options de génération, la RTL (Runtime Type Library), ce qui le rend compatible avec l'implémentation de Borland, donc avec Delphi et Turbo Pascal.

Les librairies

Free Pascal est fourni avec un ensemble de librairies permettant la génération d'applications Linux, Windows mais aussi OSX. Parmi celles-ci, il y a heaptrc qui est incluse à la compilation (option -gh) qui facilite la détection des fuites mémoires, en générant un rapport d'allocations/dé-allocations.

Capacités et options

Free Pascal supporte l'inclusion de code en assembleur dans le source ou dans des fichiers externes. Les syntaxes INTEL, AT&T sont supportées. Il est possible d'utiliser un compilateur assembleur externe (GNU AS, GNU GAS, Yasm, Nasm ...). Free Pascal supporte bien sûr l'instruction « inline », ce qui permet l'optimisation de l'exécution de certaines structures de codes, en l'incluant directement à la place de l'appel. Les instructions conditionnelles sont supportées, ainsi que les macros. FPC supporte les modificateurs de mode d'appel comme « CDECL », « export », « stdcall », « popstack ». L'édition des liens peut-être dynamique, statique ou smart, cette dernière option permettant une optimisation du code lié.

Documentations

Le site de Free Pascal contient de nombreux guides et exemples, pour les développeurs qui l'utilisent mais, aussi pour le développement. Il existe aussi un forum hébergé sur le projet Lazarus, puisque les deux sont fortement liés. Compte tenu de sa compatibilité avec Delphi, le développeur bénéficiera ou pourra profiter de l'ensemble des ressources et tutoriels disponibles.

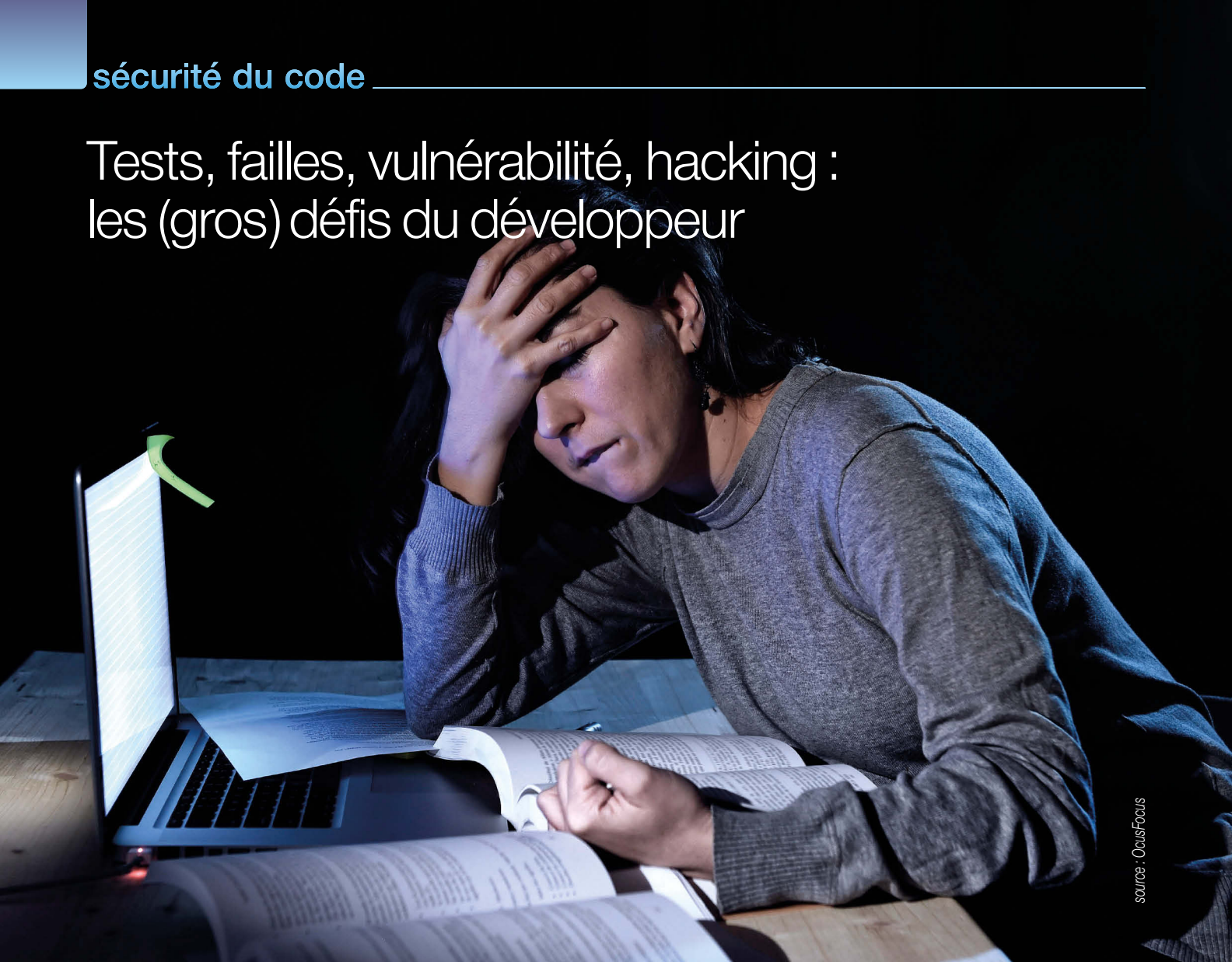
Conclusion

Un code source écrit pour Delphi sera aisément transposable sous Free Pascal, permettant ainsi un portage sous Linux ou OS X, d'une application ou d'un composant.

L'utilisation d'un environnement de haut niveau ne se justifie plus, on l'associera donc avec CodeTyphon ou Lazarus, par exemple. Pour les habitués, la transition sera aisée.

Site web : Free Pascal : http://www.freepascal.org/

Tests, failles, vulnérabilité, hacking : les (gros) défis du développeur


source : OcusFocus

Le développeur aime coder et faire du code et encore du code. Mais la qualité n'est pas forcément son point fort même si aujourd'hui, la qualité du code devient un enjeu pour les entreprises, les éditeurs. Le bug gênant peut tuer la réputation d'un jeu, d'une app mobile.

Cette qualité passe par les bonnes pratiques de développement, une approche plus agile du projet et bien entendu, l'utilisation des tests. Les tests permettent de traquer les bugs et de les résoudre, mais ils vont aussi améliorer la qualité de l'application et réduire, de facto, la surface d'attaque potentielle. Mais, les tests ne suffisent pas à supprimer toutes les failles de sécurité. Ce serait trop facile !

Il existe de très nombreuses failles de sécurité dans les systèmes, les navigateurs, les langages, les plateformes. Et là vous avez l'embarras du choix : phishing, malware en tout genre, DDoS, réseaux sociaux, injections SQL/LDAP /XML..., bugger overflows (très gros classique), authentification, système/plateforme non patché, sécurité mal réglée/configurée, des portes laissées ouvertes dans le code en production... Et personne n'est à l'abri.

LE TEST TU AIMERAS (UN PEU)

Non, ne partez pas faire votre jogging ou votre partie de Star Wars Battlefront, car les tests sont utiles. En réalité, le test recouvre des réalités multiples car il existe de nombreux types de tests : unitaires, fonctionnels, multinavigateurs, non-régression, accessibilité, performance, stress, montée en charge/charge, IHM, intégration. Dans les démarches agiles et DevOps, les tests ont une place centrale.

Gadget de bureau : Un Jenkins Lighter à l'aide d'un Arduino

Chez SAP, nous prenons la qualité du code au sérieux. C'est pourquoi, et comme pour tout développement de logiciel qui se respecte, nous mettons en place des Jenkins pour l'intégration continue du code source. En quelques mots, Jenkins reconstruit le code et exécute les tests après chaque commit. Un e-mail est ensuite envoyé avec la liste des tests qui sont en échec (entre autres).

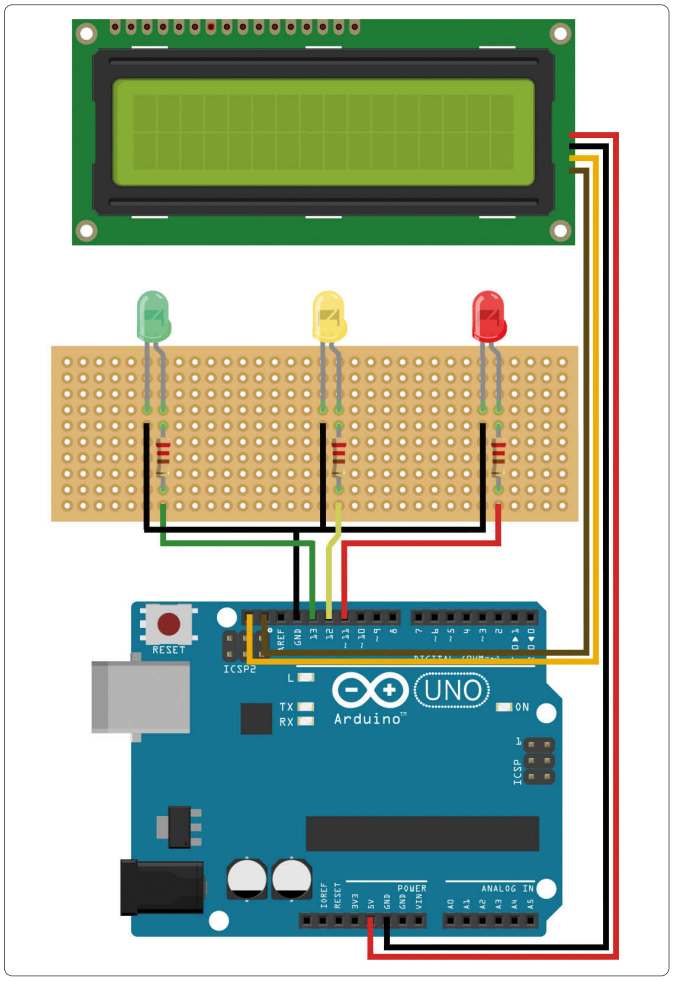
Joey Bronner - @joeybr
Développeur web chez **SAP**
http://blog.joeybronner.fr

Nous avons tous une tonne de mails (vous aussi, pas vrai ?!), il est donc facile de passer à côté de l'échec d'une build et donc de la laisser dans cet état pendant plusieurs minutes, ou des heures... Pour rendre la consultation de l'état du code plus facile et ludique, je vais détailler le montage d'un "Jenkins Lighter" à l'aide d'un Arduino et de quelques composants de base. Ce projet a donc pour but de mettre en évidence l'état actuel de la build avec 3 LEDs de couleurs rouge, jaune et verte ainsi que d'un afficheur LCD qui donne le statut de la build (succès, en cours, échec).

Pour la partie hardware, la liste du matériel utilisé :

- 1 x Arduino Uno (~3,00€)
- 1 x LCD IC2 16x2 + YWRobot LCM1602 (~2,20€)
- 3 x LEDs G/R/Y (~0,20€)
- 3 x résistances (~0,10€)
- 1 x Carte vierge de prototypage 2x8cm (~2,15€) *facultatif*

Le schéma électronique (ci-contre)



Pré-requis

IDE Arduino : https://www.arduino.cc/en/Main/Software
IDE Python (PyCharm par exemple) :
https://www.jetbrains.com/pycharm/download/
La librairie externe **LiquidCrystal_I2C** (https://goo.gl/Z7V4fz), à télécharger et à ajouter à l'IDE Arduino via le menu "Sketch > Include Library > Add .ZIP Library...".

Le sketch Arduino

Le sketch Arduino est simple, on établit une connexion via le port série à 9600 bauds... et en fonction de l'information qu'il reçoit (quatre possibilités), la platine va allumer une LED et afficher le statut correspondant à l'état de la build à l'écran. Ci-dessous, le code du programme qui est lancé à l'allumage de l'Arduino (l'ensemble des méthodes dites "Utilitaires" *(ledOn() ; ledOff() ; writeScreen() etc...)* se trouvent dans le fichier **utils.ino**).

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>

// Global variables
int incomingByte = 0;
int redPin = 13;
int yelPin = 12;
int grePin = 11;
String projectName = "YOUR-JENKINS-NAME";

LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

/*
 * Setup
 */
void setup() {
  // Screen & leds initialization
  screenInitialization();
  ledInitialization();

  // Listening on serial port (sets data rate to 9600 bps)
  Serial.begin(9600);
}

/*
```

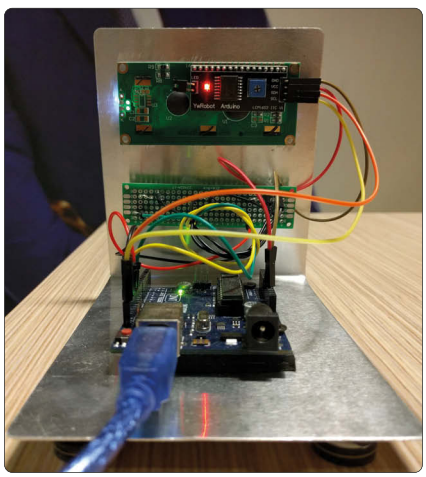
```
* Loop
*/
void loop() {

 if (Serial.available() > 0) { incomingByte = Serial.read(); }

 if (incomingByte == 'f') {
  red();
  writeScreen(1, 0, "FAILURE", false);
 } else if (incomingByte == 'b') {
  yellow();
  writeScreen(1, 0, "BUILDING", false);
 } else if (incomingByte == 's') {
  green();
  writeScreen(1, 0, "SUCCESS", false);
 } else if (incomingByte == 'u') {
  yellow();
  writeScreen(1, 0, "UNSTABLE", false);
 } else {
  screenWaiting();
  ledWaiting();
 }
}
```

Le script Python

Il faut remplacer la valeur de "JENKINS_START_URL" avec le début de l'URL de votre serveur Jenkins *(jusqu'au séparateur (/) du job)*. La liste des jobs *(oui, vous pouvez en ajouter plusieurs!)* est à mettre dans le tableau JOBS. Une fois lancé, ce script interroge en boucle l'API REST de Jenkins pour obtenir l'état actuel de la build. L'intervalle de temps de la boucle while (2 secondes par défaut) peut être modifiée selon votre envie via la variable "INTERVAL".

Le résultat est ensuite parsé et les informations nécessaires (statut, etc.) sont envoyées à l'Arduino via le port 9600.

```
# Imported libraries
import json
import sys
import urllib2
import time
import serial

# Configurations
arduino = serial.Serial('COM3', 9600)

# Global variables
SUCCESS = 's'
FAILURE = 'f'
BUILDING = 'b'
UNSTABLE = 'u'
JENKINS_START_URL = 'URL_TO_YOUR_JENKINS_JOBS'
JENKINS_END_URL = '/lastBuild/api/json'
JOBS = ['build-sbi-ui-master']
INTERVAL = 2 # Seconds

def get_status(jobName):
 # Perform Jenkins global job URL
 try:
  stream = urllib2.urlopen(JENKINS_START_URL + jobName + JENKINS_END_URL)
 except urllib2.HTTPError, e:
  print 'Error: Problem with URL: ' + str(e.code)
  sys.exit(2)

 # Parse response for global job URL
 try:
  buildStatus = json.load(stream)
 except:
  print "Error: Failed to parse JSON file"
  sys.exit(3)
 return jobName, buildStatus["timestamp"], buildStatus["result"]
while(1):
 for job in JOBS:
  status = get_status(job)
  print status[0], status[1], status[2]
  if status[2] == "UNSTABLE":
   arduino.write(UNSTABLE)
  elif status[2] == "SUCCESS":
   arduino.write(SUCCESS)
  elif status[2] == "FAILURE":
   arduino.write(FAILURE)
  elif status[2] == "BUILDING":
   arduino.write(BUILDING)
  elif status[2] == None:
   arduino.write(BUILDING)

  # Waiting time before to loop
  time.sleep(INTERVAL)
```



Le code source

Le code source complet incluant les commentaires ainsi que tous les fichiers sont disponibles et ouverts à contributeurs sur Github : http://github.com/joeybronner/jenkins_lighter

En action

Pour ma part, j'ai simplement utilisé le matériel *(LEDs, écran, etc...)* d'un kit de démarrage Arduino ainsi qu'une plaque en alu "faite maison" pour monter ce projet. Selon la folie de votre imagination, vous pouvez utiliser un feu de signalisation, feu arrière de voiture ou tout autre objet qui permet d'avoir un feedback visuel des différentes couleurs de votre Jenkins.

Pourquoi tester ?

Cet article va parler de tests, et donc de code. Parler de code est assez vague, car nous ne faisons pas tous exactement le même métier. Mais quel que soit votre domaine, vous aurez besoin de tests. Vos besoins seront toutefois sensiblement différents de ce que je décris ici.

Fabien Maury
Développeur Java, **Arolla**

De quoi parlons-nous ?

Ici je vais vous parler des tests dans mon quotidien, qui ressemble peut-être au vôtre :
Je suis un prestataire, j'arrive donc sur des projets dans diverses entreprises, sur des durées plus ou moins longues. **Mon code sera donc repris par quelqu'un d'autre. Je travaille sur des applications dites 'métier'**. Les règles que je vais devoir implémenter sont dictées par des personnes et ces règles :

- Peuvent être ambiguës ;
- Peuvent être contradictoires entre elles ;
- Vont changer.

Ce qui est normal car nous rentrons dans le domaine de l'humain. Ce type de code est le code le plus vivant que l'on puisse trouver, il va **évoluer régulièrement**, et va donc ensuite **devoir être refactoré** afin de correspondre à la meilleure implémentation possible pour cette situation. Si un code n'est pas testé, il ne peut être refactoré (ce qui arrive dans les lieux où la phrase « never touch a running system » devient loi et où l'on copie colle le code gaiement par peur de modifier l'ancien).

- *Refactoring : action de retravailler du code sans ajouter de fonctionnalité ni corriger de bugs ni induire de régression.*

Pourquoi tester ? **Pour que le code soit refactorable, sinon nous avons écrit du code Legacy.**

Bob reprend mon code

Prenons l'exemple de Bob, qui arrive après moi sur le projet, et doit ajouter une règle métier. Bob rajoute donc un test qui vérifie la règle (le test est en erreur pour l'instant), puis implémente la règle. Le test passe au vert (oui, Bob fait du Test Driven Development !) mais juste avant de commiter, au moment du build (décidément Bob est un bon gars, car il joue les tests avant d'envoyer son code !) un ancien test ne passe plus.
Premier constat : un test existe (je n'aurais effectué qu'un test manuel lors de mon développement pour vérifier mon code, Bob n'aurait pas détecté cette régression).

- *Pourquoi tester ?* **Pour vérifier que l'implémentation restera conforme à la règle attendue.**

Bob se pose donc la même question que tout le monde à cette étape : doit-il **modifier le test** ou **modifier son implémentation** ?
Si le test en question est incompréhensible :

- Nom de test vague ;
- Initialisation complexe (overdose de mock) ;
- De nombreuses assertions, nom explicite d'un point de vue métier.

Alors Bob finira sans doute, après avoir tenté de comprendre l'intention pendant 15mn, par modifier le test pour qu'il passe. Et c'est ainsi que l'on se retrouve avec des tests qui vérifient des comportements faux.
Le pire étant que faute d'avoir empêché une régression, ils vont maintenant bien vérifier que le bug est toujours présent, quitte à notifier d'une erreur un bon samaritain venu corriger l'anomalie. Est-ce le résultat que vous attendiez de vos tests ? Je ne pense pas.
Pourquoi tester ? **Parce que vos tests servent de documentation**

Comment savoir si le test est encore d'actualité ?

1) Des noms de test explicites

Un nom comme public void should_send_confirmation_sms_when_created_user_have_phone_number permet d'énoncer clairement la règle testée ici et oblige à découper vos tests par règle de gestion. Beaucoup mieux que public void *testUserCreation*

2) Des tests lisibles

Bien séparer l'initialisation, l'action testée et les assertions rendront le test plus lisible. Cela peut aller du saut de ligne à l'utilisation de la syntaxe Gherkin...en passant par des builders.
Il est alors facile pour Bob d'identifier si cette règle est encore d'actualité, ou d'aller se renseigner si cette règle précise est bien compromise par la nouvelle. Il se peut aussi que l'on ait mis le doigt sur une incompatibilité entre règles.
Le plus souvent, nous sommes notre propre Bob, mais 3 mois plus tard. Penser à Bob, c'est penser à soi.
Des tests unitaires lisibles qui documentent le code, c'est un premier pas vers de la documentation exécutable.
Après avoir statué sur ce test, Bob va pouvoir faire passer tous les tests en connaissance de cause, puis refactorer (pour introduire un pattern plus adéquat, extraire du code, etc.).
A force de refactoring, et au fil du temps, le code initial que j'avais écrit disparaîtra en grande partie pour mieux s'adapter aux nouvelles contraintes, ou juste pour mieux correspondre à la vision du nouveau développeur.

"Care and Quality are internal and external aspects of the same thing. Someone who sees Quality and feels it as he works is someone who cares. Someone who cares about what he sees and does, is someone who's bound to have some characteristics of quality."

Robert M. Pirsig,
Zen and the Art of Motorcycle Maintenance: An Inquiry Into Values

Avant tout une question de qualité

Nous testons pour nous assurer de la qualité d'une application, mais cette qualité peut prendre plusieurs visages :

- La qualité externe, qui est la qualité ressentie par l'utilisateur : elle dépend du bon fonctionnement des fonctionnalités (résultat attendu, bugs rencontrés, etc.), de l'ergonomie, de la réactivité, la compréhension de l'application (bonne ergonomie, messages d'erreurs explicites, etc.) ;
- La qualité interne, qui est la qualité de votre code : elle concerne principalement la facilité à implémenter ou modifier une fonctionnalité dans l'application. On prendra donc en compte la lisibilité de votre code, sa couverture de test, le respect des principes SOLID, la duplication de code. En bref le respect des bonnes pratiques de développement. C'est cette qualité que vous pouvez surveiller en utilisant des outils comme Sonar ;
- La "qualité" vue à travers une norme. Respect de la norme NF Z42-013 pour un produit d'archivage légal par exemple. Nous n'en parlerons pas plus car hors sujet concernant cet article.

Les qualités internes et externes sont deux aspects importants d'une application. L'une d'elle va impacter directement votre business, sera bien comprise par les managers et son feedback en production sera rapide. L'autre (la qualité interne) ne parlera souvent qu'aux développeurs (dans le meilleur des cas), et elle aura un impact direct (mais pas forcément immédiat) sur les coûts de développement, mais ne sera ressentie que trop tard côté management.

Votre application a besoin de qualité interne et externe, et pour chacune vous aurez besoin de différents types de tests.

Du “pourquoi” au “comment”

Il y a d’autres façons d’aborder ce « pourquoi ». Il est aujourd’hui communément admis qu’il est « plus propre » de tester, mais beaucoup de personnes testent sans plus vraiment se demander pourquoi au risque de perdre leur but et de dénaturer l’intention première.
Écrire un test est donc une bonne chose, mais encore faut-il qu’il soit écrit pour être utile et la seule façon d’y arriver est de se poser les bonnes questions. Cela va déjà nous permettre de savoir le type de test qu’il nous faut.
Je souhaite tester :

- Que mon validateur prenne en compte tous les cas métiers de façon exhaustive : plusieurs tests unitaires ;
- Que mon validateur utilise correctement tel service externe de référentiel : un test d’intégration
- Que mon validateur est bien appelé lors de la validation du formulaire après un click sur le bouton “OK” ou lors d’un appel sur l’API REST : un test end-to-end

Les tests unitaires

Le test unitaire va tester une partie du code indépendamment du reste, dans un contexte donné. Il va tester une règle particulière, un embranchement de code.
Ce sont les tests unitaires qui participent essentiellement à la qualité interne du code, car ils aident à séparer le code. Et du code testable est en général bien isolé et respecte SRP.
Ils ont pour vocation d’être nombreux, de s'exécuter rapidement, et d’être joués plusieurs fois par jours. Fig.1.
Ce sont les plus rapides à mettre en place, et leur rapidité d'exécution leur permet d’être exhaustifs, et de parcourir tous les cas possibles. Par contre étant donné qu’ils s'exécutent dans un “bac à sable”, leur pertinence dépend de votre compréhension du contexte, et ils ne permettent pas de garantir que l’application répond au besoin, et que la fonctionnalité fonctionne dans son ensemble.

Les tests end-to-end

Les tests end-to-end vont tester votre application, comme leur nom l’indique, de bout en bout. Ces tests passeront par le lancement de votre système/application, et dérouleront des scénarios complets en utilisant vos écrans ou vos web-services. Pour être reproductibles, l’enjeu sera d’arriver à utiliser un jeu de données stable et cohérent à chaque lancement, ce qui rend ces tests en général très coûteux en temps d'exécution (qu’ils soient manuels ou automatisés). La bonne nouvelle étant que vous aurez testé un cas particulier de votre fonctionnalité, et vous serez assurés de son bon fonctionnement. La qualité externe est donc bien vérifiée. Cependant vous pourrez rarement vous permettre d’être exhaustifs, au risque d’allonger considérablement les temps d'exécutions des tests. De plus ce sont des tests fragiles, car les risques d’un changement quelque part dans la chaîne de traitement sont plus importants. Le diagnostique en cas d’échec en sera d’autant plus laborieux car l’erreur peut provenir de n’importe quelle partie de votre application. Il est donc préférable de détecter bon nombre de bugs en amonts (grâce aux tests unitaires) Fig.2.

Fig.1 **Tests unitaires**

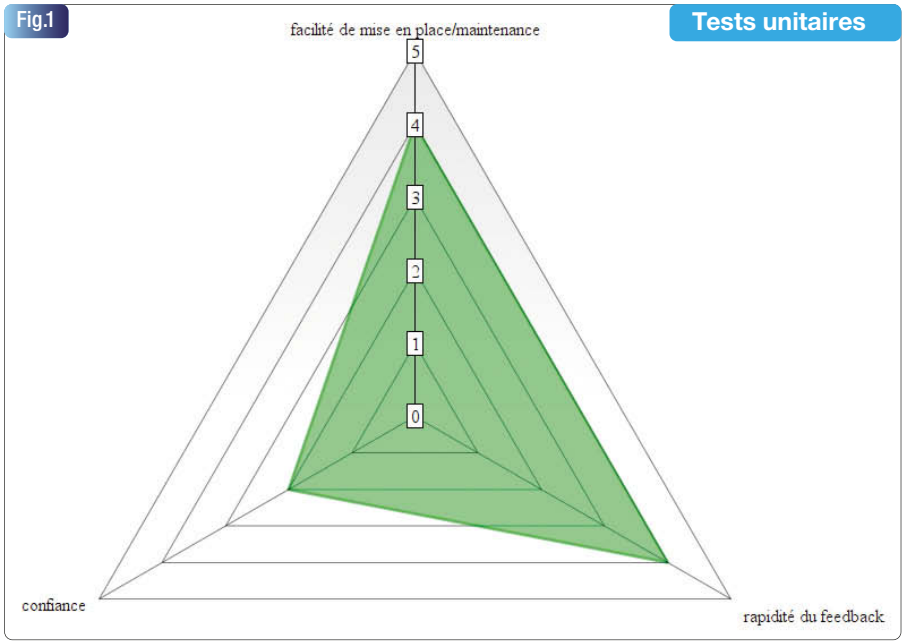
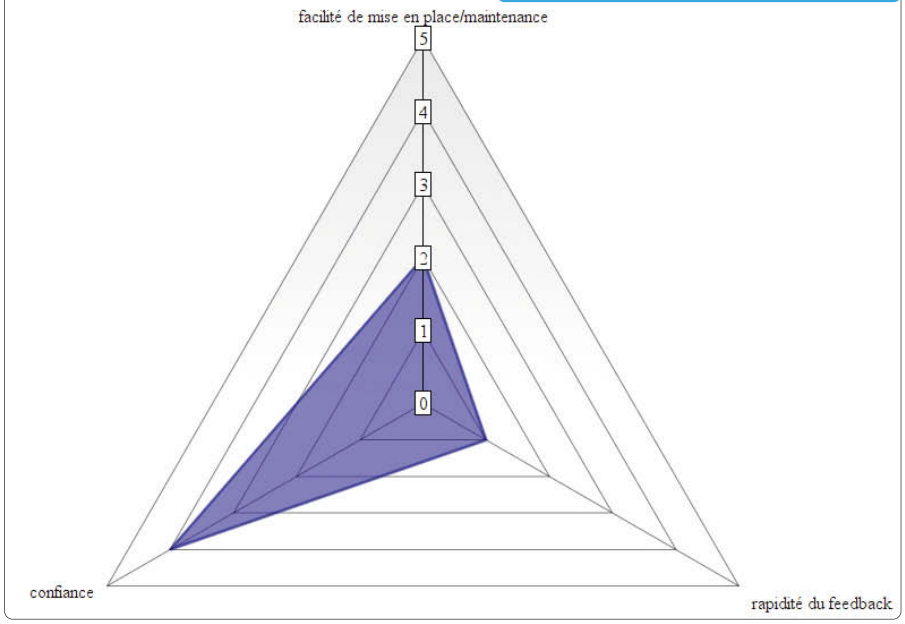


Fig.2 **Tests bout-en-bout (end-to-end)**



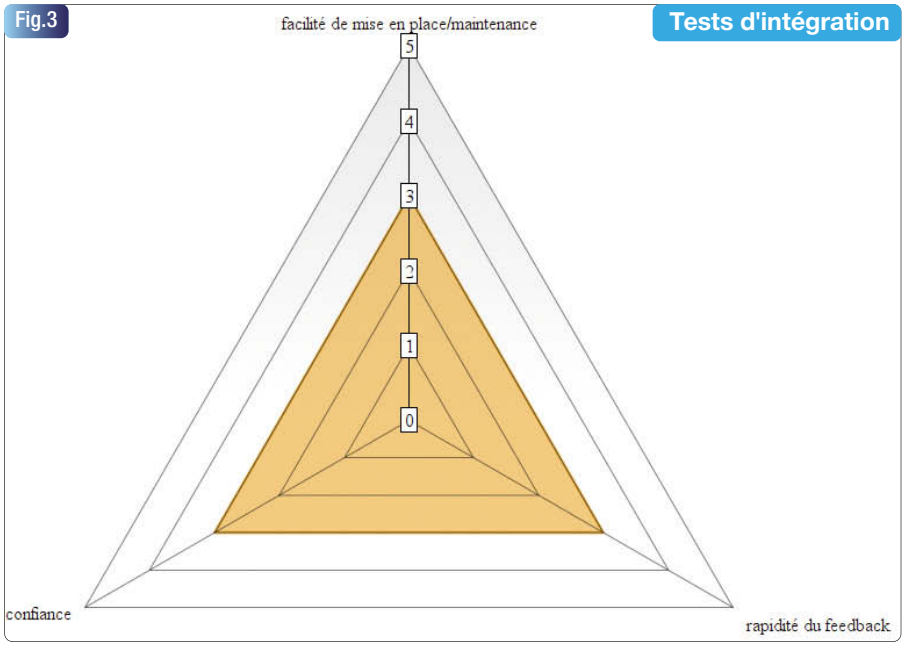
Par contre ces tests seront aussi utilisables dans le cadre des tests de performances, afin de qualifier la résistance de votre application (et/ou de votre architecture) face à la charge. Le focus sera moins sur les assertions des retours de services que sur la résistance au nombre d’appels simultanées (et/ou le temps pendant lequel l’application est stressée).

Les tests d’intégration

Attention, sur chaque projet la notion de tests d’intégration est différente. Je vais donc définir une vision du test d’intégration.
Le test d’intégration va tester qu’un de vos composants interagit bien avec un autre composant (interne ou externe). Par exemple vous souhaitez tester que vous arrivez à lancer une

coquille de batch grâce à votre listener de file AMQP, ou que vous arrivez à bien interagir avec un coffre-fort numérique via HTTP, ou encore que vos Repository récupèrent bien des données sur un schéma SQL réaliste. Pour finir, pourquoi ne pas simplement tester que votre tambouille IOC fonctionne, et que vos composants sont bien injectés. Étant donné que mocker un composant que l'on ne maîtrise pas est inutile, le test d'intégration sera souvent un bon filet au test unitaire, afin de vérifier un bout de "vrai vie" sur cette partie du code. Ils sont plus coûteux en temps, mais peuvent souvent être cantonnés à des modules qui une fois écrits, auront un cycle de vie différent de l'application et peuvent donc être extraits en dépendances (qui concernent plutôt les adaptateurs et autres connecteurs). Fig.3.

Fig.3 Tests d'intégration



D'autres perspectives ?

Nous avons classifié les tests en fonction de leur niveau de dépendance ou d'isolation, ce qui est le découpage le plus classique, mais il y a d'autres façons d'aborder le sujet.
Par exemple le **BDD (Behavior driven development)** où le test n'est plus seulement outil de validation, mais support de discussion et véritable espace de collaboration entre les métiers de développeurs, testeurs et les experts du fameux métier. De plus, les tests servent ici de documentation exécutable.
Autre exemple le **property-based-testing**, qui au lieu de se baser sur les habituels schémas "pour tel input X, j'aurai un output Y" se base sur des invariants, en prenant en entrée des données aléatoires. Ce type de test est très utile sur un environnement complexe où l'exhaustivité des cas métiers est une chimère non atteignable. Par exemple, il est impossible de tester unitairement tous les nombres pairs, par contre on peut tester qu'une règle vraie pour un chiffre impaire est validée sur un chiffre aléatoire multiplié par deux. Cet article est très orienté "tests automatisés", car oui je pense que toute tâche manuelle est une perte de temps, et l'investissement sera perdu dès lors que le code aura été modifié. Cependant cette assertion est fausse dans le cas des **tests exploratoires**, qui laisse plus de place à l'autonomie du testeur et à l'improvisation. En effet des plans de tests répétitifs (ou des tests automatisés) seront des "ennemis connus" des développeurs, qui risquent de se focaliser sur les "cas testés". Cette discipline est un bon complément pour éviter la monotonie et mettre en avant des failles dans le code ou des soucis d'ergonomie. Si ce sujet vous intéresse, je vous conseille d'aller faire un tour sur le blog de James Marcus Bach, et de vous intéresser au **context-driven testing** et au **rapid software testing**. Tout cela pour dire que le test est un vaste sujet, et nous ne ferons donc ici que l'effleurer, chaque sorte de test méritant l'écriture d'un livre (a minima).
Cependant j'espère que cela vous aidera à orienter vos recherches.

Quels tests choisir ?

Vous aurez remarqué que je ne semble privilégier aucun de ces types de tests (intégrations, unitaires ou bouts-en-bouts), et en effet rien ne sert de se lancer dans une guerre de clochers, car vous aurez besoin de chacun d'eux. Il y a cependant un constat simple et éprouvé, qui veut que le feedback sur la santé de votre application doive être le plus rapide possible : il faut donc de tout, mais surtout des tests unitaires.
Je ne l'ai pas inventé, c'est ainsi qu'est pensée la pyramide des tests : Fig.4. Il faut donc surtout éviter de tout tester en end-to-end, ce qui fige votre feedback, mais aussi le "tout test unitaire" certes rapide, mais qui n'empêche pas la qualité externe de se dégrader. Un autre fléau étant des tests pénibles à maintenir. Votre ennemi sera surtout le mauvais test, celui qui est contre-productif, qui vous apporte plus de soucis et de temps perdu que de valeur.

Penser à l'échec

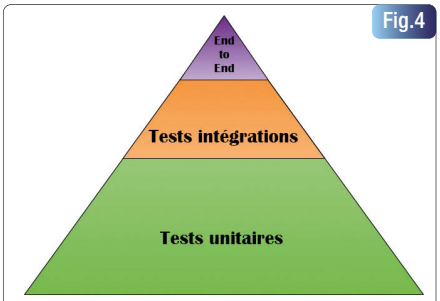
Si l'on écrit des tests, c'est pour les voir échouer, mais encore faut-il que cet échec soit utile :

- Le test doit échouer pour les bonnes raisons ;
- La raison de l'erreur doit être claire et explicite ;
- Aidez votre futur diagnostique : contexte d'exécution, règle testée, etc.

Tester mes tests ?

Mes tests sont souvent du code, et tout code devrait être testé... Mais comment tester mes tests ? En fait il existe une méthode pour "tester" vos tests ; cela s'appelle le mutating testing. Cela consiste à jouer vos tests unitaires sur des variantes de votre code : des "mutants".
En effet le framework de mutating testing va inverser des conditions booléennes, remplacer des ET par des OU, enlever des instructions (etc., etc.) de votre code de production ....
Avant de lancer vos tests. Et là, normalement, vos tests doivent échouer. A savoir que le concept est intéressant, mais cela génère un énorme flot de données à analyser pour déterminer la qualité réelle de vos tests. Vos tests étant testés, la boucle est ainsi bouclée.

Fig.4



Les tests, nos vrais livrables ?

On peut remarquer que du code écrit à un instant t, le code de production ne persistera plus qu'en partie, alors que la plupart des tests seront encore d'actualité. C'est pour cette raison que je préfère souvent considérer que je livre à mon client des tests, et non du code de production. Ce dernier n'étant, au final, qu'un détail d'implémentation des règles de gestion demandées par le client.
Donc je conclurai par : pourquoi tester ? Parce que votre code de test a plus de chances de survivre que votre code de production !

Générer ses tests pour une solution plus robuste avec Smart Unit Test

Avec les nouvelles méthodologies de développement, les livraisons plus fréquentes, et une exigence de la qualité toujours à son maximum, les tests sont devenus l'une des clés de la réussite d'une solution de qualité, maintenable et évolutive.

Jérémy LANDON
Consultant
Infinite Square

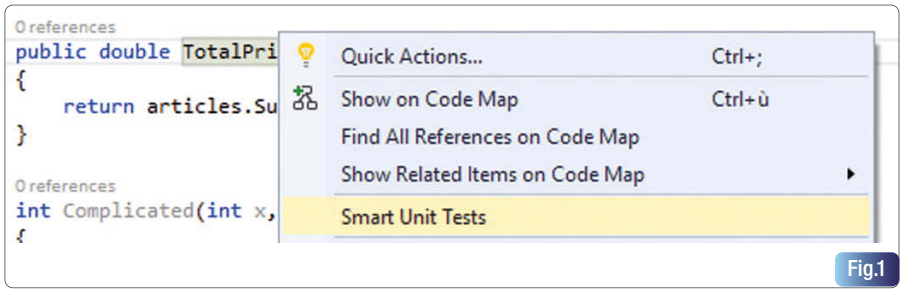


Fig.1

D'un point de vue technique et dans un monde idéal, afin de réduire le nombre de bugs, tous les membres de l'équipe devraient être au courant des évolutions d'un logiciel, se souvenir de chaque interaction dans le code pour éviter les régressions, documenter tout travail et surtout lire les documentations des autres... en bref il est totalement utopique de penser que cela peut fonctionner ne serait-ce que sur le court terme.

Pour pallier ce problème et obtenir une solution robuste, il faut obligatoirement passer par les tests techniques (entre autres les tests unitaires). Ce système permet de spécifier le code et fournir implicitement un support de référence.

Les tests tout le monde veut en faire, mais ils sont malheureusement trop peu réalisés, les raisons de cet échec sont multiples : méconnaissance des bénéfices, « manque de temps » (qui est un faux problème), coût sur le court terme, l'espérance qu'ils seront réalisés plus tard...

Afin d'aider à l'adoption des tests il devient important comme dans tout domaine de s'outiller. Le développement .NET ne manque pas de framework de tests, pour ne citer que les plus connus : MSTest, NUnit, MbUnit ou encore xUnit.

Néanmoins ces boîtes à outils pour les développeurs à elles seules ne suffisent parfois pas à l'adoption des tests.

Un test dans le fond c'est assez simple : il vérifie un comportement :

- Ce comportement peut avoir un côté fonctionnel : aucun outil ne peut encore remplacer un humain sur ce point;
- Ou avoir un côté technique, comme par exemple tester les limites d'une méthode, mais encore faut-il avoir un aperçu de ces limites.

Une puissante analyse du code

Microsoft a sorti avec Visual Studio 2015 une nouvelle fonctionnalité nommée Smart Unit Test. Cette fonctionnalité est un assistant de développement de tests unitaires, qui aide le développeur à trouver des bugs précoces et réduit le coût de maintenance des tests.

Cette nouvelle fonctionnalité est aussi connue sous le nom de Pex, en effet Microsoft a importé dans Visual Studio 2015 cet utilitaire réalisé à l'origine par Microsoft Research sur la génération de tests unitaires, utilisable dans les versions ultérieures par extension Visual Studio.

Comment cela fonctionne ?

Smart Unit Test repose sur un moteur d'analyse de code permettant de détecter de manière optimale tous les chemins possibles d'exécution d'une méthode. Pour faire simple il va pouvoir générer une série de tests unitaires permettant de tester les limites d'une méthode. Ici aucun niveau fonctionnel, mais que de la technique :

- Un paramètre null est-il géré ?
- Une exception est-elle levée si un paramètre vaut 0, sa valeur maximale, ou une valeur aléatoire ?
- Passer en paramètre un objet dont les propriétés sont assignées à null entrave-t-elle l'exécution du code ? Etc.

Les limites d'une méthode sont parfois extrêmement complexes à déterminer, une méthode peut en effet se complexifier par l'appel d'autres méthodes, elles-mêmes possédant leurs propres chemins d'exécution, boucles et récursivités. Tout cela génère un nombre infini de combinaisons.

Plus qu'un outil d'aide, Smart Unit Test ne se résume pas à reproduire une logique qu'un développeur pourrait avoir, il est capable de détecter des possibilités qui seraient impossible à détecter par un développeur car trop nombreuses ou trop complexes. Bien entendu la force de Smart Unit Test est de savoir déterminer la limite utile du nombre de possibilités, ou plutôt de pouvoir borner l'exécution du code.

A l'utilisation Smart Unit Test va réaliser plusieurs actions :

- Découvrir dynamiquement toutes les branches d'exécution possibles d'une méthode;
- Synthétiser les valeurs d'entrée à appliquer afin d'exécuter ces chemins;
- Enregistrer les sorties d'une méthode cible à la suite de l'exécution de la méthode;
- Persister les tests sous la forme de tests unitaires (code) afin d'obtenir une haute couverture de test et garder ces analyses utiles pour les futures évolutions du programme.

A l'utilisation ça donne quoi ?

L'utilisation est très simple : clic droit sur la méthode -> Smart Unit Test. Fig.1.

L'analyse se met en route, et le moteur teste tous les chemins utiles. Une nouvelle fenêtre fait son apparition, nommée simplement Smart Unit Tests. Elle affiche dans un premier temps les résultats de l'analyse :

- Tous les cas de test réalisés;
- Les valeurs de chaque paramètre;
- Le résultat;
- La potentielle exception levée;
- Rouge ou vert, si le test a été concluant ou non. Fig.2.

Après cette analyse il est possible de convertir tout cas de test en test unitaire. Il suffit pour cela de **sélectionner les cas voulus** -> **Clic Droit** -> **Save**. Smart Unit Test génère automatiquement un projet de test accompagné de deux fichiers :

- Une classe de test contenant un appel simple vers la méthode à tester : NomDelaClasseCible**Test**.cs;
- Une classe de test contenant l'ensemble des cas de tests utilisés par Smart Unit Test : ce dernier est référencé en tant que dépendance au premier et sera généré durant la compila-

tion : NomDelaClasseCible**Test.NomDeLaMethodeCible.g**.cs.

Chacun des cas testés est simplement représenté sous la forme de méthode de test, classiquement supplanté par l'attribut **TestMethod.** Fig.3.

Une fois les tests générés, il est possible de les rejouer comme n'importe quel test via **Test Explorer**. Pour un cas aussi simple, Smart Unit Test a été capable de générer les tests de manière automatique et ainsi de pouvoir aider à obtenir un code plus robuste. Dans la vue **Event,** Smart Unit Test propose une analyse sur le code, et sur la réalisation des tests.

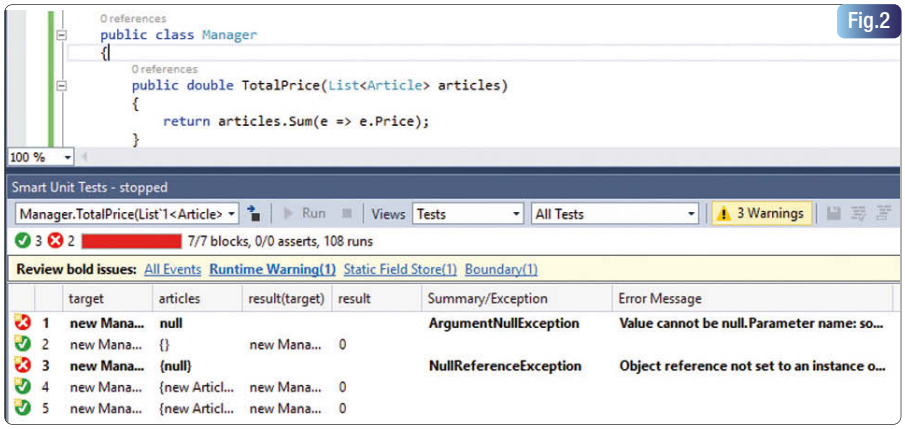
L'outil peut proposer certaines optimisations architecturales pour rendre le code plus facilement testable : générer automatiquement des factory, supprimer la redondance, rendre plus pertinent le typage... Une information remontera souvent, présent dans l'onglet **Boundary**, elle permet de redéfinir la limite des bornes du test.

Dans le cadre d'une méthode ne contenant pas beaucoup de boucles, pas de méthode récursive ou très peu de valeurs à tester, l'analyse est très rapide. Mais ce genre de méthode ne reflète pas la majeure partie des cas présents en développement, il est donc important de borner le moteur de Smart Unit Test.

Smart Unit Test doit-il générer tous mes tests ?

NON. Smart Unit Test n'est pas un outil ayant pour objectif de remplacer la phase de test, mais c'est un outil complémentaire aidant à gagner du temps et à fiabiliser cette étape. Smart Unit Test est particulièrement utile dans certaines situations :

- **Tester le comportement déjà en place** : dans le cas de code inconnu sans aucune couverture de test, dans lequel il va falloir interagir avec différents composants sans pour autant obtenir de la régression. Il faudra donc s'assurer que nos modifications n'affecteront pas négativement le comportement des méthodes. Smart Unit Test aura ici pour rôle de figer le comportement rapidement afin de nous garantir un certain scénario et de garantir une stabilité du code au vu de nos futures modifications.
- **Etablir les bases du test** : tester une méthode est souvent long, tant les cas sont nombreux, Smart Unit Test permet de libérer le temps de développement des tests de base (test d'exception, de limite) qui peuvent représenter une grande partie de la couverture de la méthode, afin de se concentrer sur des tests qui ne peuvent pas être automatisés : à savoir les tests ayant un certain niveau fonctionnel
- **Réduire le temps de maintenance des tests** : dans le cas où les tests sont réalisés par un développeur, lors du changement d'une méthode testée, il faudrait repasser sur chacun des tests pour les adapter à la nouvelle architecture. Grace à Smart Unit Test, il suffit simplement de modifier le code, les méthodes de tests seront automatiquement générées.



Quelles sont les limites ?

Smart Unit Test est avant tout un outil d'aide et possède plusieurs limites :

- Pas de gestion du multi-threading;
- L'analyse peut être réalisée sur n'importe programme .NET, néanmoins Smart Unit Test n'est capable de générer que du C#;
- Le moteur de test peut déterminer des cas de valeurs numéraires pertinentes, dans le cas par exemple ou une méthode cible prend en paramètre une série de nombres. Néanmoins l'analyse connait des limites lors de nombres à virgule flottante.

Conclusion

Face aux cycles de développement toujours plus courts et intenses et des exigences toujours plus élevées, chaque heure gagnée est un cadeau qui ne se refuse pas. Avec le boom de la méta-programmation, il est maintenant établi que la productivité et la qualité passent par la génération automatique : faire mieux avec moins.

Smart Unit Test est un outil formidable pour obtenir une application sous test rapidement, simplement et efficacement, même si celui-ci ne remplacera pas totalement le travail d'un développeur il sera d'une aide très précieuse. Cette fonctionnalité est incorporée nativement dans Visual Studio 2015, en revanche pour ceux voulant en profiter dans les autres versions, il est toujours possible d'utiliser PEX.

TESTEUR : UN VRAI MÉTIER

Testeur est un vrai métier, avec des formations et des profils. Les entreprises peuvent créer des cellules qualités spécifiques.

Parmi les profils que l'on peut trouver sur le marché :

- testeur
- analyste des tests
- analyste technique des tests
- gestionnaire des environnements de tests
- consultant tests
- chef de projet de tests

N'hésitez pas à vous renseigner, notamment auprès du Comité Français du Test Logiciel.

La rédaction

Le TDD comme rempart contre nos biais

Le développement piloté par les tests ou Test Driven Development - TDD - a changé notre relation au code. Profondément. Pour le meilleur. Plus détendus, plus efficaces et plus pertinents, les professionnels que nous sommes devenus le doivent beaucoup à cette manière de coder qui agit comme une force de rappel contre de nombreux biais ou difficultés dont nous sommes parfois victimes en tant que développeurs. Nous allons ici zoomer sur quelques-uns de ces biais qui faisaient partie de notre quotidien de développeur pour pouvoir vous expliquer en quoi la pratique du TDD nous a permis de les dominer.

Bruno BOUCARD - @brunoboucard
À la fois, Coach Agile et Software Craftsman, il enseigne le TDD, le Refactoring, le BDD et DDD. Il est aussi spécialiste sur les technologies Microsoft depuis de nombreuses années. Bruno est aussi président de la société 42 SKILLZ.

Thomas PIERRAIN - @tpierrain
Créateur de la librairie open source NFluent (http://www.n-fluent.net/), architecte technique à la Société Générale et adepte du DDD, des pratiques XP et du Software Craftsmanship, Thomas pratique le TDD depuis plus de 10 ans maintenant.

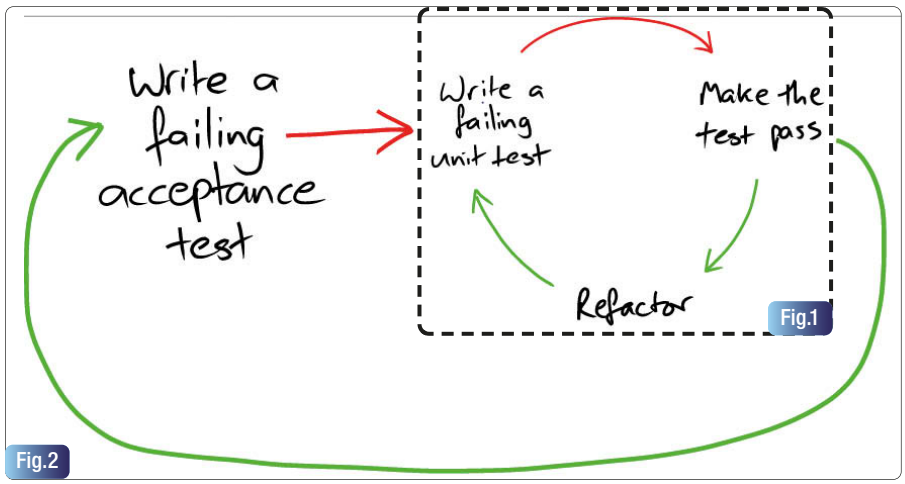


Schéma de la double boucle de l'outside-in TDD (Nat Pryce)

Plus détendus

Thomas : Pour commencer, je dois vous faire une confession : même si je me soigne, j'ai une fâcheuse tendance à la procrastination... Dit autrement, il m'arrive assez souvent de remettre au lendemain ce que je peux faire le jour même. En général cela se produit quand je tombe sur quelque chose qui me dérange (ouvrir des factures, faire les magasins pour les cadeaux de Noël ;-). J'ai alors tendance à tourner autour du pot en m'autorisant parenthèse sur parenthèse, avant de me pencher sur la corvée quand je n'ai vraiment plus le temps d'y échapper. Pour le développeur que j'étais au début de ma carrière, cela se traduisait par de nombreuses digressions que j'opérais lorsque j'étais face à un problème ou à un DEV qui m'apparaissait compliqué à résoudre, compliqué à attaquer. Il m'arrivait alors dans ces moments de tomber dans une situation similaire à l'angoisse de la page blanche : par quel angle dois-je attaquer ce développement ? Qu'est-ce qui est le plus important dans ce contexte ? Ai-je suffisamment fait le tour de la question pour être pertinent dans mon DEV ? Bref, il m'arrivait souvent de tourner autour du pot en googlant sur des sujets connexes et en passant de page en page avant de me retrouver pris par le temps - et d'être contraint d'attaquer le développement, un peu plus stressé du coup (car sans rabe coté délais). Le TDD lui nous décomplexe en nous forçant à attaquer notre développement par **une première fonctionnalité, la plus simple possible** (en fait par le test pour celle-ci). Nous sommes même encouragés à utiliser des valeurs en dur dans les premières étapes de l'implémentation. Cette ambition un peu paralysante de faire les choses au mieux, à la cible dès le début donc, disparaît. Cette façon de développer en commençant par un test et qui nous pousse mécaniquement à faire des petits pas (RED-GREEN-REFACTOR... RED-GREEN-REFACTOR...) me permet du coup de réaliser facilement la chose la plus difficile lorsqu'on est victime de procrastination : COMMEN-CER. Ce premier petit test est tout simple, sans ambition ; il agit sur moi comme un déclencheur. C'est en effet le petit coup derrière l'épaule dont j'ai besoin pour commencer et me mettre en mouvement (oui parce qu'une fois qu'on est lancé il n'y a plus de procrastination qui tienne, c'est le principe).

Bruno : Un autre intérêt de cette approche test-first, c'est le caractère motivant du workflow RED-GREEN-REFACTOR. Pour rappel :

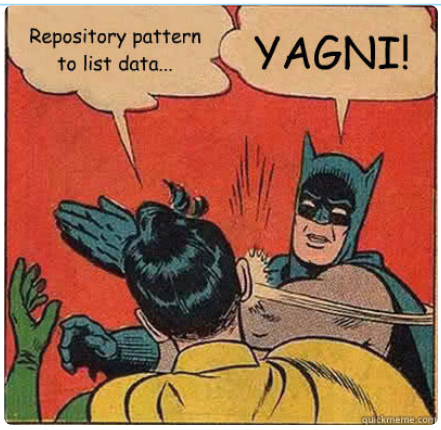
- **RED**: c'est la première étape. On commence par identifier le comportement ou la fonctionnalité qu'on souhaite rajouter et on écrit un test qui illustre celle-ci. Bien sûr, comme l'implémentation à ce stade n'est qu'un "walking skeleton" qui émerge depuis le test (aidé de notre IDE préféré, on crée la structure vide de cette implémentation au moment où l'on définit son usage, c'est à dire depuis le code du test), le test échoue. D'où la couleur rouge. Les Anglo-Saxons résument cette étape par la formule: "Make it fail".
- **GREEN** : c'est la seconde étape qui vise à implémenter le plus rapidement possible (et c'est important de s'y tenir) le code qui est ciblé par notre test pour le faire passer au vert. Pour les Anglo-Saxons, c'est le : "Make it works". Il est important d'aller vite ici, et il est même recommandé de prendre quelques raccourcis comme des valeurs en dur qui nous aideront juste à faire passer le test le plus vite possible avant de passer à l'étape d'après (c'est très perturbant au départ, mais bénéfique, nous allons voir pourquoi en détaillant l'étape suivante).
- **REFACTOR** : c'est la troisième et dernière étape, trop souvent oubliée ou sacrifiée dans un quotidien parfois stressant. Maintenant que nous avons une implémentation et que le test passe, il s'agit ici de revoir la structure interne de notre code pour corriger tous les raccourcis pris lors du "Make it work". C'est d'ailleurs pour cela que les Anglo-Saxons parlent ici de "Make it better". Le fait d'avoir séparé les préoccupations de

coder quelque chose qui fait le job (definition-of-done précisée par le test), et ensuite d'améliorer la qualité de l'implémentation mène à une plus grande efficacité dans notre action (tant il est parfois facile de se perdre dans des optimisations à tiroirs lorsqu'on tente de faire les 2 en même temps). Fig.1.

Bon, on vient de détailler ces 3 étapes, mais en quoi ce workflow est-il encourageant ? Eth bien je ne sais pas pour vous, mais moi je trouve que les journées d'un développeur sont truffées de feedbacks négatifs... Ça commence par notre compilateur qui passe son temps à nous remonter des erreurs, à notre chaîne d'intégration continue ensuite, qui passe au rouge de temps en temps (perturbant notre cadence en nous forçant à la corriger toute affaire cessante), quand ce n'est pas le chef de projet qui se retrouve sur notre dos parce qu'il est stressé (ou parce qu'il a une bonne raison de l'être). Dans ces journées où les feedbacks négatifs sont légion et impactent sans doute plus notre mental qu'on veut bien le reconnaître. Dans ces journées donc, chaque petit test qui passe, chaque loupiotte verte du RED-**GREEN**-REFACTOR sont autant de petites victoires, de petits signes encourageants qui m'indiquent que ça avance. Je les prends comme des petites friandises qui viennent cadencer mes baby steps et ma journée.

Plus efficaces

Thomas : On vient de le voir, le TDD me permet de démarrer mes développements plus rapidement; enlevant au passage une partie inhérente du stress que je pouvais éprouver à la fin de mon travail et qui était liée à ma procrastination. Un autre bénéfice du TDD est qu'il me permet d'être plus rapide et plus efficace quand je code, ce qui est souvent contre-intuitif pour ceux qui n'ont pas l'habitude de le pratiquer. Comment est-ce possible ? Eh bien c'est lié au fait que le TDD - lorsqu'on est bien équipé - nous permet de ne pas répéter encore et encore les mêmes gestes fastidieux, les mêmes tests à la main, les mêmes sessions de débogage ou d'essais-erreurs comme j'avais l'habitude de le faire par le passé. L'équipement dont je parle ici est un outil comme NCrunch (en .NET) ou infinitest (en Java), même si je connais surtout NCrunch. Le concept : lorsque vous codez, l'outil compile, exécute les tests et analyse le test coverage en tâche de fond pour que vous puissiez avoir un feedback quasi instantané dans votre IDE (sous la forme de puces de couleur pour chaque ligne de code dans la marge: vert quand la ligne est couverte par des tests, rouge quand la ligne est impliquée dans au moins un test qui échoue, et noir lorsque la ligne de code n'est couverte par aucun test). Que vous soyez en train de refactorer du code legacy ou en train de travailler sur un tout nouveau code, cela vous permet d'avoir un feedback immédiat pour chacune de vos interventions. Un must!



Du coup, **les moments où vous avez besoin de lancer votre debugger deviennent de plus en plus rares** ce qui n'est pas pour me déplaire tant le lancement du debugger me paraît lent comparé au feedback rapide de NCrunch (plus proche d'un REPL(1) en termes de feedback).

Quelqu'un a écrit une fois sur Twitter :

Ecrire du code : 1 heure / Fixer les bugs : 4 heures

Ecrire des tests : 1 heure / Ecrire du code : 1 heure / Fixer des bugs : 15 minutes

Ca résume assez bien l'efficacité du TDD je trouve, même si l'écriture des tests et du code sont entrelacés.

Bruno : De mon côté, le TDD, m'a également permis de changer ma vision sur le développement logiciel. Effectivement, le développement traditionnel débute généralement avec une forme de sketching au niveau du code. On comme avec de la pâte à modeler, on recherche une solution technique acceptable. Après une brève prise en compte du besoin utilisateur, le développeur écrit directement quelques classes qui semblent rassurer sa compréhension. Dans les faits, ce mode encourage une forme de design qui part du code pour adhérer au besoin.

L'approche TDD est dirigée par le besoin utilisateur pour déboucher sur une proposition technique **en parfaite adéquation avec le besoin ni plus ni moins**. En d'autres mots, on doit s'interroger en termes de cas d'utilisations portés par les comportements attendus par l'utilisateur avant d'engendrer son code. C'est un point important.

Avec ce workflow RED-GREEN-REFACTOR, le TDD est également l'assurance d'avancer par petit pas en toute quiétude.

Pas d'effet tunnel

Votre code est « commité » régulièrement, il est donc facile de comprendre la démarche d'implémentation a posteriori. Le code est ultra simple, et ne contient aucune astuce, un autre développeur peut le reprendre sans risque d'ambiguïté sur son intention.

Plus pertinents

Thomas : Le TDD s'il est pratiqué en mode "**outside-in**", se révèle diablement efficace **pour construire sans hésitation ni gâchis, le bon système**. Comme je n'ai pas encore eu l'occasion d'introduire ce terme, laissez-moi vous le présenter.

De nos jours, on distingue deux formes majeures de TDD: l'école classique, et l'école dite de Londres (ou "outside-in TDD"). La forme classique est celle introduite par le créateur du TDD: Kent Beck (aussi le créateur de l'eXtreme Programming). Dans cette forme historique également la plus connue, on part en général du centre du système pour construire autour et faire grossir celui-ci par strates successives pour arriver au système final. L'école de Londres elle (appelée ainsi, car elle a été introduite par Steve Freeman et Nat Pryce dans leur ouvrage de référence "Growing Object-Oriented Software Guided by Tests" - le GOOS), propose de considérer le système à construire d'un point de vue extérieur, comme une grosse boîte noire que nous allons définir, remplir et implémenter petit à petit.

La double boucle de l'outside-in TDD

Alors au début bien sûr, cette boîte noire est vide. Et nous allons définir petit à petit ses contours et la façon d'interagir avec elle via des tests "grosse maille" : des tests d'acceptance qui viseront le système dans son ensemble (la boîte noire). Sur le chemin, nous allons aussi être amenés à écrire des tests unitaires qui vont nous aider à coder petit à petit l'intérieur de la boîte en partant des interfaces définies par les tests d'acceptance. On appelle ça la démarche de la "double boucle". Fig.2.

Bruno : Détaillons un peu : tout commence par une 1ere boucle de RED-GREEN-REFACTOR au niveau acceptance (le sujet à l'étude est donc à cet instant le système dans son ensemble). Le RED nous permet de définir, à travers un 1er test d'acceptance, une des fonctionnalités de cette boîte noire. Le test d'acceptance échouant, nous descendons tout de suite d'un niveau d'abstraction pour commencer à itérer sur plein de petits RED-GREEN-REFACTOR au niveau unitaire, pour implémenter ce qu'il faut à l'intérieur de la boîte ; l'objectif étant toujours de faire passer notre 1er test d'acceptance. Une fois que c'est fait, on continue et on réalise l'étape REFACTOR du test d'acceptance initial... Avant

(1) *Read-Eval-Print Loop*

de repartir pour un nouveau tour de grande boucle en commençant par écrire le second test d'acceptance, qui nous fait redescendre au niveau des petites boucles unitaires pour son implémentation et ainsi de suite...

Une approche minimaliste et YAGNI par essence

Thomas : L'intérêt de l'outside-in est d'**empêcher qu'on se perde en route en implémentant des choses qui n'ont pas un lien direct avec le résultat final** (i.e. notre système pris dans sa globalité). Dès le départ on est piloté par la définition des contours et des interactions avec celui-ci (l'approche classique pouvant nous mener à découvrir trop tard ce qu'on s'est loupé et qu'on n'a pas construit le bon système ou un système pratique à utiliser).

Aujourd'hui, je peux dire que je pratique presque exclusivement l'outside-in TDD (à l'exception de certains petits katas de code où il m'arrive de refaire du TDD classique).

Cela nous ramène à une caractéristique importante du TDD : l'efficacité. Une efficacité liée au minimalisme de la démarche qui n'est rien d'autre que l'application du principe YAGNI (*You Ain't Gonna Need It*) : ici on ne code rien qui ne soit directement lié à un test, et donc si on s'y prend bien, à une fonctionnalité requise pour notre système.

Au final des professionnels plus épanouis, tout simplement

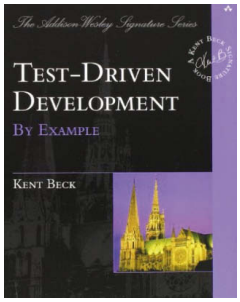
Thomas : M'ayant permis de régler définitivement certains problèmes qui m'empêchaient d'être réellement efficace en tant que développeur (procrastination, debugging intempestif, effet tunnel lors de la réalisation, hors sujets dans certaines sessions de DEV où on finit par se faire plaisir en oubliant l'objectif initial ...), la pratique du TDD a fait de moi un meilleur développeur, mais surtout un professionnel plus aguerri, capable désormais de me concentrer sur ce qui est le plus important pour mes clients ou mes utilisateurs finaux sans me perdre en route.

Vers un développement logiciel en harmonie avec soi même

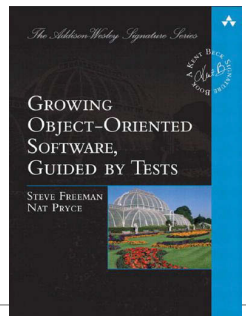
Bruno : Si vous êtes développeurs, vous avez sans doute une vie ponctuée de périodes calmes où vous développez gentiment les premières briques de votre projet, puis des périodes plus intenses lorsque la date de la première mise en production va se rapprocher, pour finir par des périodes très intenses où les problèmes de productions vont dominer votre quotidien. Ce n'est pas vraiment une vie simple à la fois pour vous, et votre entourage personnel. Vous pouvez être dans le déni et considérer ce mode de vie comme une fatalité, ou bien vous pouvez changer de mode de développement. La pratique du TDD dépasse largement les aspects techniques, car elle permet d'objectiver son métier de manière itérative. À l'instar d'un footing où l'objectif n'est pas de courir le plus vite possible, mais de trouver son rythme, le TDD s'inscrit dans un rythme cadencé par les tests. Thomas en a parlé, on choisira toujours un ordre de difficulté croissant. Les premiers tests seront simples et permettront de « croquer » le début d'implémentation ». Dans ce début de parcours, les tests seront relativement génériques et le code assez spécifique. Au fil du chemin parcouru, les tests seront de plus en plus spécifiques et le code de plus en plus générique, car au fil des implémentations les refactorings successifs, vont apporter leurs lots de simplifications. Ceci pour obtenir un code qui transpire les comportements fonctionnels à travers les noms des tests que vous avez choisis, permettant d'être à la fois lisible et facile à maintenir. Il peut arriver que votre parcours s'étale sur plusieurs jours. Mais comme chaque test développé doit donner lieu à un commit libellé par le comportement que vous avez illustré, vous gagnez une forme d'assurance vis-à-vis de votre travail. Avec le temps, vous gagnerez en confiance sur votre métier et vous serez plus rassurés sur votre code. C'est le début d'une attitude plus sereine et plus en harmonie avec vous-même. Il vous restera même du coup plus d'énergie pour aller vers les autres (vos collègues, vos utilisateurs, vos clients, etc).

Après ce 1er article consacré au "pourquoi" du TDD, nous vous proposerons par la suite d'autres articles (illustrés par du code cette fois) sur les pièges et les écueils à éviter lorsqu'on fait du TDD, parce qu'il est très facile de se louper, et de passer du coup à côté de cette pratique incroyable. Ce serait dommage. Vraiment...

Quelques références sur le TDD



Le premier ouvrage sur le TDD est celui de Ken Beck. Il représente la première incarnation de cette pratique. La communauté qualifie la méthode décrite dans ce livre, de TDD classic. Le second ouvrage est plus abouti, il décrit l'usage du TDD dans toutes les phases de développement d'un projet. C'est de ce livre qu'est tiré le schéma de la double-boucle de l'outside-in. La communauté qualifie la méthode décrite dans ce livre, d'*école de Londres* en hommage aux auteurs du livre.



Tester la qualité de votre sécurité avec ZAP

Avant la mise en ligne de votre site ou de votre application, vous devez obligatoirement tester la sécurité de celui-ci pour détecter, rechercher des failles ou des vulnérabilités éventuelles. Cette étape rentre dans le processus des tests de qualité et vous pourrez mesurer votre qualité de sécurité avec l'outil Zed Attack Proxy (ZAP).

Christophe Villeneuve
Consultant IT pour Neuros, Mozilla Reps, auteur du livre "Drupal avancé" aux éditions Eyrolles et auteur aux Éditions ENI, Rédacteur pour WebRIVER, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), Drupagora...

Avertissement : L'utilisation détournée de cet outil peut être sanctionnée par le Code pénal R323-1 et R323-5. L'auteur de l'article et le magazine Programmez ne peuvent pas être tenus pour responsables

Récompensé à plusieurs reprises depuis son apparition en 2010, ZAP a pour but d'effectuer des tests de pénétration. Il peut être utilisé à n'importe quel moment dans un cycle de développement par l'ensemble des acteurs du projet (développeurs, chef de projet...). Depuis, il a trouvé sa place dans ce secteur de la sécurité et il est important de vous en parler aujourd'hui.

ZAP Attack !

Cet outil doit être utilisé dans un but précis : tester votre application. Il est conçu pour être utilisé par des personnes ayant une vaste expérience en matière de sécurité, et, en tant que telle, il est idéal pour les développeurs et les testeurs fonctionnels qui sont nouveaux dans les tests de pénétration.

ZAP est un outil simple, open source, développé et maintenu par la communauté visant à rendre le monde plus sûr, représentée par OWASP (Open Web Application Security Project).

L'outil est utilisable sans avoir de grandes connaissances au niveau de la sécurité applicative. Il fournit des scanners automatiques ainsi qu'un ensemble d'outils qui vous permettent de trouver des failles de sécurité manuellement. Ainsi, cet outil va répondre à certaines fonctionnalités :

- Explorer votre application, comme si vous téléchargiez toutes les pages Web pour les regarder hors ligne ;
- Le scan passif navigue dans l'application et il détecte automatiquement s'il y a des problèmes éventuels de sécurité ;
- Le scan actif s'occupe de lancer des attaques directement à l'instant T ;
- Vous pouvez l'utiliser manuellement pour faire des tests bien précis ou sur une évolution ;
- La possibilité de scripter pour tester une fonctionnalité métier bien précise et sensible.

Par ailleurs, il répond aux fonctions de tests standards, qui sont les failles les plus courantes comme l'authentification (Identifiant / Mot de passe), l'ajout de certaines règles dans les formulaires, les connexions utilisateurs avec LDAP... Une des fonctionnalités évoluées de l'outil permet de le placer dans le processus des tests de qualité qui rentre dans le cycle de l'intégration continue.

De plus, il est possible de l'intégrer avec d'autres outils comme Jenkins, Sélénium... et de le déclencher à chaque livraison d'un build. ZAP fonctionne sur Linux, Windows, Mac et est disponible à l'adresse suivante :

https://github.com/zaproxy/zaproxy/wiki/Downloads

Scan actif

Le scan actif est la version la plus rapide à mettre en place, car vous saisissez l'adresse de votre site Internet et vous patientez. Fig.1.

Après quelques minutes d'attente, vous obtenez une synthèse des alertes éventuelles.

Comme le montre l'image 1, de nombreuses alertes ont été repérées dont certaines sont critiques, comme ici des failles d'injections SQL. Fig.2. Pour connaître le détail, vous devez sélectionner la ligne souhaitée pour obtenir les informations. Ainsi vous obtenez les URL qui posent problème. Fig.3. Le logiciel propose une description de la faille en Français et la méthode d'attaque utilisée concernant l'injection SQL. Fig.4.

Pour corriger cette faille, l'outil offre la solution. Tout d'abord une explication détaillée du problème, et la référence associée. Celle-ci aborde en détails la faille repérée et propose la solution adaptée, quel que soit votre langage de développement. Fig.5.

Scan passif

Contrairement au scan actif, le scan passif ne modifie pas les réponses venant du serveur. Il regarde seulement les réponses afin d'identifier les vulnérabilités. Pour cela deux méthodes existent pour effectuer ce type de tests.

La première méthode consiste d'utiliser ZAP comme proxy : il se positionne entre le naviga-

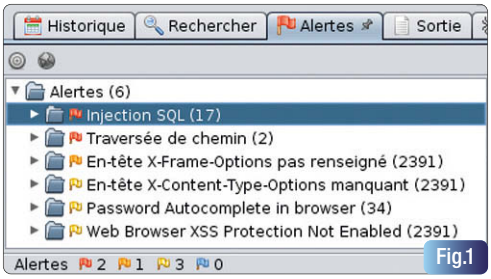


Fig.1

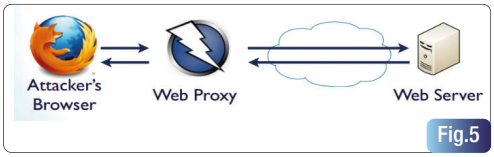


Fig.5

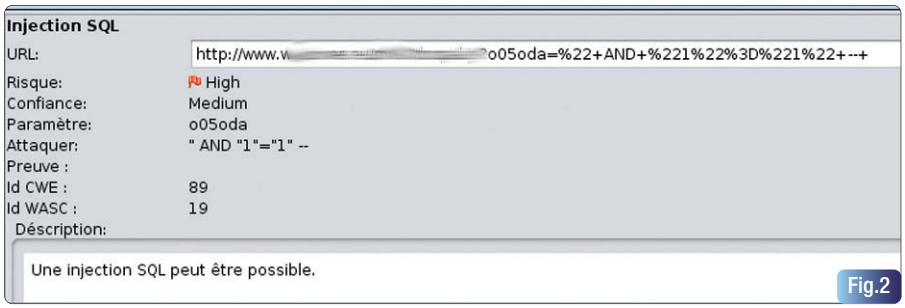


Fig.2

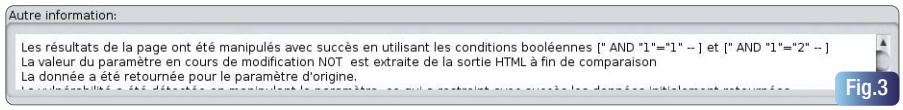


Fig.3

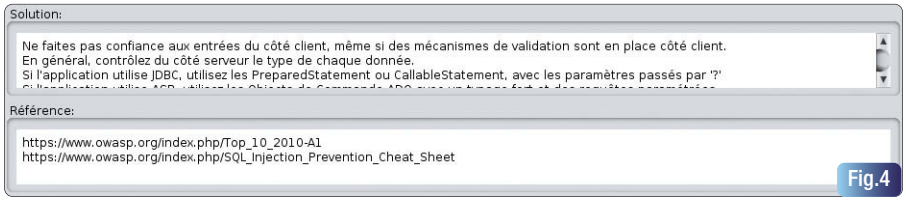


Fig.4

teur et le serveur. Fig.6. Cette technique permet de suivre la navigation d'une personne dans votre application et de détecter automatiquement les problèmes éventuels. Il va mémoriser la méthode de navigation pour mémoriser l'envoi des requêtes et des réponses.

L'autre méthode du scan passif consiste à balayer les réponses GET/POST de vos pages, pour en rechercher les informations comme :

- Les en-têtes 'Header' HTTP incomplètes ;
- L'identification des XMLHttpRequest mal configurée qui sont souvent associées avec les échanges de données en JavaScript ;
- La saisie semi-automatique des mots de passe dans les navigateurs ;
- Une authentification faible ;
- Les éventuels CSRF (Cross Site Request Forgery).

Pour exécuter un scan passif, vous paramétrez un proxy manuellement, accessible à partir du menu réseau de l'onglet Avancé des préférences du navigateur Firefox. Nous utilisons l'adresse 127.0.0.1 et le port 8080. Fig.7. L'étape suivante consiste à paramétrer le proxy de l'outil ZAP en spécifiant la même adresse que le navigateur. Fig.8. Pour arriver à l'écran de configuration, vous passez par le menu outils \ options et proxy local. La dernière étape consiste à enregistrer le scénario, ou la nouvelle fonctionnalité mise en place dans votre projet, c'est à dire, à mémoriser les requêtes et les réponses. Fig.9.

Lorsque le scénario est terminé, vous pouvez consulter et voir les différentes requêtes et réponses jouées. Fig.10.

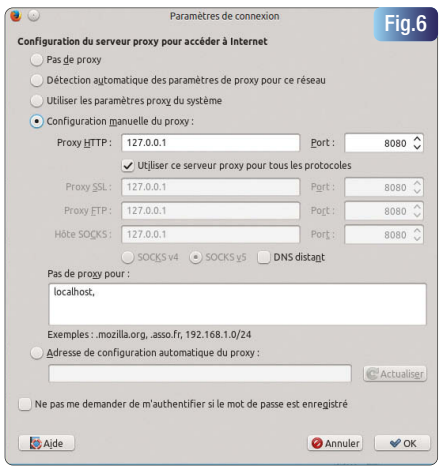


Fig.6

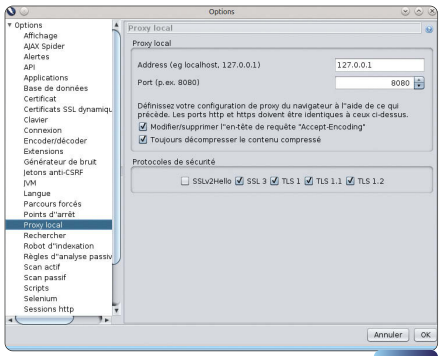


Fig.7

De nombreux indicateurs sont présents, comme les informations POST, HTTP, Cookie, Title, Content-length... Ces informations, si elles ne sont pas correctement remplies seront des mines d'informations pour les personnes mal intentionnées.

Résultat

Quel que soit le mode de balayage (actif ou passif), un rapport complet vous sera mis à disposition avec les différents niveaux de menace associés à la vulnérabilité : haut, moyen, bas.

Les résultats d'analyses sont utilisés pour générer le rapport. Il peut se sauvegarder et être exporté dans un fichier HTML ou XML, dans le but de ne pas faire l'ensemble du script régulièrement. Fig.11.

Le mode avancé

Quel que soit le type du balayage, vous pouvez analyser en profondeur une page Web ou un dossier. Pour cela vous devez sélectionner la ligne avec un clic droit pour obtenir différents types d'attaques.

Ainsi, vous trouverez la possibilité d'émettre des attaques supplémentaires dans des dossiers, de surveiller en détail la navigation des utilisateurs... L'article ne rentrera pas plus en détail dans ce mode avancé, car il est trop sensible.

Outils

ZAP propose de nombreux outils supplémentaires et variés.

Un scanner de port

Cette fonction scanne les ports ouverts du projet. Le résultat permet de connaître les ports ouverts non utilisés et qui peuvent poser des problèmes.

Fuzzing

Le fuzzing est le processus d'envoi d'informations invalides, ce qui permet de cacher le mode d'attaque. Il permet d'observer le comportement de l'application.

Hash

La fonctionnalité Hash permet d'encoder et de décoder un texte saisi.

Extensions

De nombreuses extensions sont disponibles. Elles peuvent être activées à volonté pour tester LDAP, la fixation de session, Webservices, langage... La liste est très longue, mais disponible à l'adresse suivante : https://github.com/zaproxy/zap-extensions


Fig.8

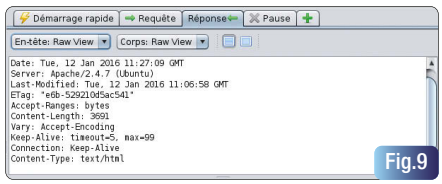


Fig.9

Scripts

Il est aussi possible d'ajouter des scripts qui vont répondre spécifiquement à vos problèmes métiers. Chaque page peut bénéficier de tests approfondis et s'appuie sur les langages standards (PHP, JavaScript, Python, Ruby, etc..) ou le langage Zest script.

Liens utiles

OWASP Zed Attack Proxy Project :
https://www.owasp.org/index.php/ZAP
Télécharger ZAP :
https://github.com/zaproxy/zaproxy/wiki/Downloads

Conclusion

Au fil des années, les attaques sont de plus en plus fréquentes, c'est pourquoi les tests de sécurité applicatifs ne doivent pas être pris à la légère ; ils sont malheureusement souvent oubliés ou supprimés dans le cheminement d'un projet pour respecter les délais de conception et de réalisation. Il est indispensable d'en prendre conscience et de sensibiliser cette étape dans la validation de votre produit ou de votre site, car les résultats peuvent être désastreux si cette étape n'est pas effectuée. Grâce à l'outil ZAP, vous allez connaître le niveau exact de sécurité de votre application et l'intégrer dans le processus de développement et de tests. De plus, lorsque vous mettez en place un système d'intégration continue, il est nécessaire de refaire les tests de sécurité jusqu'à l'environnement de pré-prod (image de la prod). Par ailleurs, les mises à jour sont régulières, car il veut vous aider à améliorer la détection des vulnérabilités qui sont de plus en plus présentes, et à détecter les nouvelles méthodes de pénétration.

.

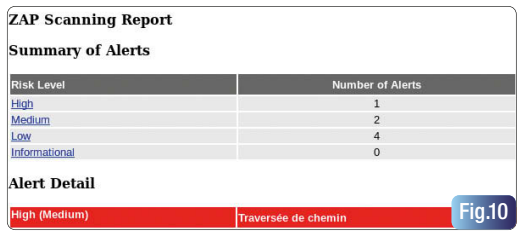


Fig.10

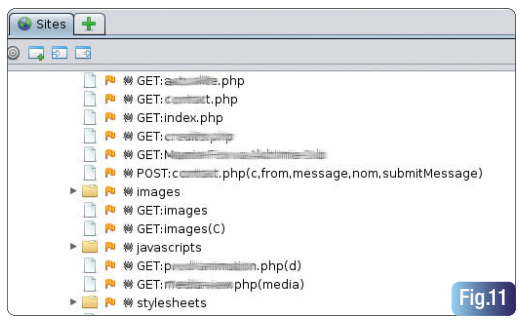


Fig.11

Implémenter des tests unitaires sur du code existant

Nous allons dans cet article parler des tests unitaires : tout d'abord les situer dans la taxonomie des tests, puis savoir comment les implémenter. Nous mettrons ensuite l'accent sur l'importance d'avoir des tests véritablement unitaires. Enfin, nous verrons comment modifier du code existant pour le rendre testable et testé.

Christophe HERAL @ChrisHeral
Artisan Logiciel - **Neotech Solutions**

Définition des tests unitaires

On peut regrouper les différents types de tests en 4 catégories, selon le fait qu'ils soient plutôt techniques ou fonctionnels, plutôt dans les cas standards ou aux limites du système : Fig.1.

Conformément à la **pyramide des tests** de Mike Cohn, les tests de base et les plus importants à mettre en place sont les tests unitaires, soit ceux du quadrant 2 dans la typologie ci-contre. Fig.2.

(image tirée de www.mountaingoatsoftware.com)

Il faut tout d'abord bien identifier le **système sous test** (SUT : System Under Test), en général une classe. Un test unitaire doit vérifier le bon comportement du SUT.

On retrouve 3 étapes dans un test unitaire que l'on peut résumer par l'acronyme **AAA** :

- **Arrange** : créer les conditions d'exécution du test et préparer les éléments requis en entrée ;
- **Act** : invoquer l'action que l'on souhaite tester ;
- **Assert** : vérifier le résultat de l'action et faire échouer le test si le résultat n'est pas celui attendu.

On utilise l'acronyme **FIRST** pour définir les 5 caractéristiques d'un bon test unitaire :

- **Fast** (Rapide) : c'est la seule garantie que le test soit lancé fréquemment ;
- **Independent** (Indépendant) ou **Isolated** (Isolé) : un test ne doit pas établir les conditions d'exécution du test suivant ;
- **Repeateable** (Reproductible) : à chaque exécution, vous devez toujours obtenir le même résultat, et ce dans n'importe quel environnement ;
- **Self-Validating** (auto-validant) : ils réussissent ou ils échouent (pas de journal à consulter) ;
- **Timely** (au moment opportun) : c'est-à-dire juste avant le code de production qui permet de les réussir.

Quand réaliser ses tests ?

Historiquement, les tests unitaires étaient réalisés après le développement. Au lieu de cela, l'Extreme Programming propose de faire l'inverse : écrire son code de test **AVANT** d'écrire son code de production, technique appelée **TDD : Test Driven Development**. Comme l'explique Dan North, le développeur doit alors réfléchir à son code en partant d'un autre bout de code qui va l'utiliser, ce qui favorise la **conception émergente**. La conception est réalisée au fil de l'eau au fur et à mesure que les abstractions recherchées apparaissent dans le code déjà écrit et testé. On évite donc toute complexité accidentelle et tout code inutile : si tous les tests passent, inutile de rajouter du code. Faire du TDD demande seulement de respecter **3 lois** :

1. Pas de code de production tant que vous n'avez pas écrit un test qui échoue ;
2. Écrire uniquement un test suffisant pour échouer (l'impossibilité de compiler est un échec) ;
3. Écrire uniquement le code de production suffisant pour faire passer le test qui échoue.

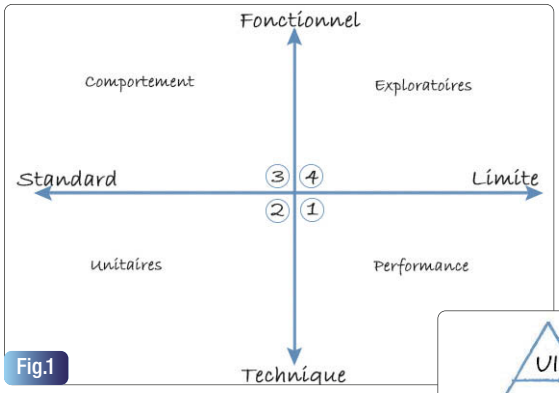


Fig.1

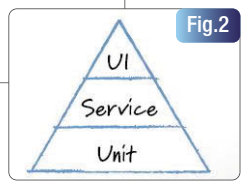


Fig.2

Le cycle doit être très court. Par ailleurs, chaque nouveau bug est l'occasion d'écrire un test couvrant ce cas d'utilisation.

Pour conclure, n'oublions pas cette citation d'Uncle Bob : "Un test propre a 3 caractéristiques : **Lisibilité, Lisibilité et Lisibilité**".

Comment les implémenter ?

Pour écrire ces tests, nous allons utiliser un **framework de tests unitaires**. Il en existe de nombreux dans tous les langages de développement. On peut citer par exemple **JUnit** en Java, NUnit, XUnit.Net ou MSTest en C#, **Jasmine, QUnit ou Mocha** en JavaScript, **PHPUnit** en PHP...

Tous les frameworks de la famille **xUnit** sont ressemblants. Dans la suite de l'article, nous utiliserons **NUnit** que l'on peut aisément référencer dans les projets Visual Studio à l'aide d'un package NuGet par exemple.

Le framework de test se base sur des attributs qui sont positionnés sur les éléments de langage :

- La classe qui contient les tests est décorée de l'attribut [**TestFixture**] ;
- La méthode de test est décorée de l'attribut [**Test**].

D'autres attributs optionnels existent. Par exemple :

- [SetUp] : la méthode ainsi décorée sera exécutée avant chaque test de la classe ;
- [TearDown] : la méthode ainsi décorée sera exécutée après chaque test de la classe ;
- [TestFixtureSetUp] (ou [OneTimeSetUp] en NUnit 3) : la méthode ainsi décorée sera exécutée avant le premier test de la classe ;
- [TestFixtureTearDown] (ou [OneTimeTearDown] en NUnit 3) : la méthode ainsi décorée sera exécutée après le dernier test de la classe ;
- [Category] : permet de ranger les tests dans des catégories distinctes ;
- [Ignore] : précise que ce test doit être ignoré lors de l'exécution des tests.

Le framework de tests va nous offrir un ensemble **d'assertions** permettant de vérifier le résultat de l'exécution.

Si une assertion échoue, la suite de la méthode n'est pas exécutée et une erreur est signalée. Il est donc préférable de tester une seule assertion au cours d'un test.

Chaque méthode peut également prendre un paramètre optionnel. Lorsque l'assertion échoue, ce paramètre permet de préciser le message d'erreur associé à l'exécution du test.

NUnit fournit un riche ensemble d'assertions en tant que méthodes statiques de la classe Assert. Ces assertions peuvent être effectuées de 2 manières :

- Avec le **modèle classique** : chaque affirmation proposée par le langage s'exprime par une méthode distincte ;
- Avec le **modèle basé sur les contraintes** : une seule méthode est utilisée *Assert.That*. La logique liée à l'assertion est intégrée dans l'objet de contrainte passé comme second paramètre à cette méthode.

Ce 2e modèle, considéré comme plus fluide ("**fluent**") tend à être de plus en plus répandu et pourrait remplacer à terme le modèle classique.
Exemple : je veux tester que "Hello" est présent dans maChaîne.
Avec le modèle classique :

```
StringAssert.Contains("Hello", maChaîne, "Hello n'est pas présent dans maChaîne")
```

Avec le modèle basé sur les contraintes :

```
Assert.That(maChaîne, Does.Contain("Hello"), "Hello n'est pas présent dans maChaîne")
```

Sous NUnit 3, certaines assertions proposées par le modèle par contraintes n'ont pas d'équivalent dans le modèle classique.
Exemple :

```
int [] array = new int [] {1, 2, 3};
Assert.That(array, Has.Exactly(1).EqualTo(3));
Assert.That(array, Has.Exactly(2).GreaterThan(1));
Assert.That(array, Has.Exactly(3).LessThan(100));
```

Des tests vraiment unitaires

L'un des travers que l'on trouve souvent en relisant des tests unitaires provient du fait que l'on teste une classe avec ses dépendances, ce qui n'en fait finalement qu'un test d'intégration.
En effet, dans ce cas, si le test échoue, on est incapable de savoir si l'échec provient du code de notre classe ou de sa dépendance.
Par définition, un test unitaire doit être totalement **indépendant** du reste de l'environnement. Notamment, il ne doit pas dépendre :

- D'une API externe ou d'une autre classe de service ;
- D'une base de données ;
- De données de production ;
- Des fichiers ou ressources du réseau ;
- Du jour et de l'heure ;
- De l'exécution d'un autre test.

L'isolation de notre SUT est une exigence fondamentale pour effectuer nos tests unitaires.
Lorsque notre SUT interagit avec d'autres collaborateurs du système, nous allons simuler ces dépendances pour rendre le test unitaire.
Au moment du test, au lieu d'utiliser son collaborateur réel, nous allons lui injecter une autre implémentation qui aura le comportement voulu.
Nous appellerons doublure un objet simulacre qui imite un vrai objet dans le but de tests. On spécifie à la doublure quels sont les appels que l'on attend et leurs valeurs de retour.
On vérifie que les appels ont bien eu lieu (et éventuellement leur ordre).
En fait, il y a 2 manières de vérifier qu'une interaction avec une dépendance s'est bien passée : :

- **La vérification de l'état** : on détermine si la méthode exécutée a fonctionné correctement en examinant l'état du SUT et de ses collaborateurs après l'exécution de la méthode ;
- **La vérification du comportement** : on détermine si la méthode exécutée a fonctionné correctement en vérifiant si les bons appels de méthode ont été effectués sur ses collaborateurs.

Selon les besoins de notre test, nous pouvons donc utiliser différents types de doublure :

- **Dummy** (fantôme) : sert à remplir un espace réservé requis pour passer le test, mais qui n'est pas utilisé par le SUT.
- **Fake** (substitut) : implémentation détournée mais qui marche (on stocke par exemple dans un fichier au lieu d'une base de données)
- **Stub** (bouchon) : renvoie les valeurs prédéfinies ("qui vont bien") au SUT.
- **Mock** (simulacre) : permet de contrôler les appels et ainsi de vérifier un comportement
- **Spy** (espion) : enregistre les sorties indirectes de l'objet pour vérification ultérieure par le SUT.

On peut les résumer dans le tableau ci-dessous :

Type de doublure	Avec configuration	Avec implémentation manuelle
Vérification d'état	Stub	Fake
Vérification de comportement	Mock	Spy

Les frameworks de mocking

L'écriture d'objets de type mock à la main peut être longue et fastidieuse. De plus, des objets peuvent contenir des bugs comme toute portion de code.
Un **framework de mocking** (aussi appelé **framework de simulacre**) permet la création automatique de toutes les variantes de doublures de tests. Ils permettent de spécifier le comportement que doit avoir l'objet mock :

- Les méthodes invoquées : paramètres d'appels et valeur de retour ;
- L'ordre d'invocations de ces méthodes ;
- Le nombre d'invocations de ces méthodes.

Ils permettent de créer dynamiquement des objets mocks, généralement à partir d'interfaces. Ils proposent également d'autres fonctionnalités :

- Simulation de cas d'erreurs en levant des exceptions ;
- Validation des appels de méthodes ;
- Validation de l'ordre de ces appels .

Dans l'exemple que nous allons détailler dans la suite de cet article, nous allons utiliser le framework de mocking *Moq*.

Rajouter des tests sur du code legacy

En relisant du code existant sur un projet, nous sommes souvent confrontés à ce que l'on appelle du "**legacy**". Cela correspond à du code qui a mal vieilli, écrit à une époque où les tests unitaires étaient beaucoup moins répandus et en méconnaissant les méthodes d'ingénierie.
On tombe alors dans un cercle vicieux : comme ce code a été écrit sans test, il est souvent difficilement testable unitairement. Il fonctionne mais comporte souvent de nombreux travers qui y sont liés :

- Les classes sont fortement couplées ;
- Certaines classes ou méthodes sont très grosses et ont trop de responsabilités.

Écrire du code propre et testable

En suivant la définition de l'Extreme Programming (Uncle Bob notamment), un **code** est **propre** lorsqu'il respecte les critères suivants (dans cet ordre) :
1. Passe tous les tests ;
2. N'est pas redondant ;
3. Exprime clairement les intentions du programmeur ;
4. Minimise le nombre d'entités (classes et méthodes).

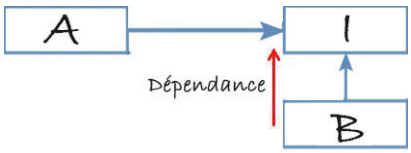
Souvent, notre code ne respecte pas ces règles. Pour aller dans ce sens, nous allons le travailler pour rendre ce code plus lisible, plus maintenable et plus évolutif, c'est ce que l'on appelle le "**refactoring**". Cela va notamment nous permettre de :

- Limiter la complexité d'une classe ;
- Supprimer la duplication de code ;
- Simplifier un algorithme ;
- Renommer des classes ou des méthodes ;
- Supprimer du code mort.

Nous pourrons par exemple pour cela appliquer un certain nombre de principes (SOLID, KISS, YAGNI, DRY, ...).
Les tests sont le filet de sécurité qui va nous permettre d'effectuer ce refactoring sereinement, sans régression sur le code existant. Pour cette raison, les tests vont donc être indispensables.
Comme décrit précédemment, notre plus grand danger concerne les dépendances sous toutes leurs formes : instanciations directes, code statique, singletons, héritage (lui préférer la composition), ...
Un design pattern va grandement nous aider pour tester notre SUT sans ses dépendances : c'est l'**inversion des dépendances** que l'on peut résumer par le principe d'Hollywood : « *Ne nous appelez pas, nous vous appellerons.* »
Au lieu d'avoir une classe A qui dépend d'une classe B, A va dépendre seulement d'une interface qui est notamment implémentée par B. Ainsi, nous aurons le choix de l'implémentation !

Sans injection de dépendances :

Avec injection de dépendances :



Exemple de refactoring

On va supposer dans notre exemple l'utilisation d'une architecture en couches classiques (UI ou host de service, logique métier, accès aux données). A l'aide de l'injection de dépendances, la couche métier n'appelle pas directement l'implémentation de la couche d'accès aux données mais une interface de celle-ci. Nous pourrons alors la tester en faisant abstraction de cette dépendance : le framework de mocking va nous permettre de simuler cet accès aux données avec un simulacre qui fonctionne tout le temps. Voici le code initial de notre classe d'accès aux données et de notre classe métier qui la consomme :

```
public class VehiculeDataAccess
{
    public Vehicule GetVehicule(int vehiculeId)
    {
        var vehicules = GetAllVehicules();
        return vehicules.FirstOrDefault(_ => _.VehiculeId == vehiculeId);
    }

    private List<Vehicule> GetAllVehicules()
    {
        var jsonString =
  File.ReadAllText(@"D:\Programmez\VehiculeSample\Vehicule.json");
        var vehicules = JsonConvert.DeserializeObject<List<Vehicule>>(jsonString);
        return vehicules;
    }
}

public class VehiculeBusiness
{
    public Vehicule GetVehicule(int vehiculeId)
    {
        var vehicule = new VehiculeDataAccess().GetVehicule(vehiculeId);
        vehicule.Chassis = string.Format("{0}-{1}", vehicule.Chassis1, vehicule.Chassis2);
        return vehicule;
    }
}
```

Le contenu du fichier *Vehicule.json* est le suivant :

```
[
  {
    "vehiculeId": "1",
    "chassis1": "123",
    "chassis2": "456"
  }
]
```

Le code de test initial est le suivant :

```
[TestFixture]
public class VehiculeBusinessTest
{
    [Test]
    public void GetVehiculeTest()
    {
        // ARRANGE
        var vehiculeBusiness = new VehiculeBusiness();
        // ACT
        var vehiculeRecupere = vehiculeBusiness.GetVehicule(1);
        // ASSERT
        Assert.That(vehiculeRecupere.Chassis, Is.EqualTo("123-456"));
    }
}
```

Cependant, l'on s'aperçoit que lorsque l'on teste notre couche métier, on se retrouve lié au code de notre couche d'accès aux données et même au contenu du fichier JSON. Si ce fichier n'est pas présent dans le chemin spécifié ou que son contenu a été modifié, le test qui vérifie le format du châssis va échouer. Pourtant le contenu de cette classe n'a pas bougé et correspond à ce qui est attendu. Finalement, ce qui nous intéresse, c'est que la couche d'accès aux données nous expose une méthode qui prend en paramètre l'identifiant du véhicule et qui renvoie le véhicule.
Par exemple :

```
public interface IVehiculeDataAccess
{
    Vehicule GetVehicule(int vehiculeId);
}
```

Notre classe *VehiculeDataAccess* l'implémente déjà. Il n'y a qu'à préciser l'interface :

```
public class VehiculeDataAccess : IVehiculeDataAccess
```

Maintenant, nous allons pouvoir injecter l'interface de la couche d'accès aux données au lieu de l'implémentation concrète :

```
public class VehiculeBusiness
{
    private readonly IVehiculeDataAccess vehiculeDataAccess;
```

```
    public VehiculeBusiness(IVehiculeDataAccess vehiculeDataAccess)
    {
        this.vehiculeDataAccess = vehiculeDataAccess;
    }

    public Vehicule GetVehicule(int vehiculeId)
    {
        var vehicule = vehiculeDataAccess.GetVehicule(vehiculeId);
        vehicule.Chassis = string.Format("{0}-{1}", vehicule.Chassis1, vehicule.Chassis2);
        return vehicule;
    }
}
```

Notre test va maintenant pouvoir utiliser un simulacre de la classe d'accès aux données qui renverra les données que l'on souhaite.

```
    [Test]
    public void GetVehiculeTest()
    {
        // ARRANGE
        var vehiculeDataAccessMock = new Mock<IVehiculeDataAccess>();
        vehiculeDataAccessMock.Setup(_ => _.GetVehicule(1)).Returns(new Vehicule
        {
            VehiculeId = 1,
            Chassis1 = "123",
            Chassis2 = "456"
        });
        var vehiculeBusiness = new VehiculeBusiness(vehiculeDataAccessMock.Object);
        // ACT
        var vehiculeRecupere = vehiculeBusiness.GetVehicule(1);
        // ASSERT
        Assert.That(vehiculeRecupere.Chassis, Is.EqualTo("123-456"));
    }
```

Notre nouveau test est plus robuste et n'est plus dépendant de l'environnement. On teste maintenant le contenu de la méthode de la couche métier et uniquement celle-là.

Conclusion

Nous avons donc mis en exergue l'importance des tests unitaires dans le développement avec les 3 étapes pour les écrire (AAA) et les 5 caractéristiques d'un bon test (FIRST). En faisant du TDD par exemple, les tests nous permettent aussi d'améliorer la conception de notre logiciel en la faisant émerger puis en faisant du refactoring pour obtenir du code propre. Nous avons vu comment les implémenter avec un framework de tests comme NUnit. Pour isoler complètement notre système sous test, nous pouvons appliquer plusieurs techniques comme l'injection de dépendances ou l'utilisation de frameworks de mocking.

Enfin, par le biais d'un exemple, nous avons modifié du code legacy pour injecter un simulacre au lieu de la classe d'accès aux données pour tester uniquement le contenu de notre couche métier.

Maintenant, il ne vous reste plus qu'à en rajouter sur vos projets si vous n'en faites pas déjà !

La sécurité : une carrière pour vous ?

La sécurité informatique est régulièrement citée comme une compétence recherchée par les entreprises. De nombreux profils existent. Parmi les profils, nous retrouvons le responsable de sécurité informatique, le fameux RSSI.

Il a pour mission de mettre en échec les attaques (de toute nature), les intrusions, les codes malveillants dans les systèmes d'une entreprise. Il établit les procédures et protocoles. Il définit les accès et les règles. Pour sa formation, il suivra des cursus sécurités, hackings, réseaux. Selon la fiche du CIDJ, le salaire moyen mensuel brut peut varier de 3 500 à 4 600 €.

L'ANSSI recense différents profils liés à la sécurité, à la cybersécurité :

Profil	Fonctions
Technicien support	Responsable de diverses activités de support, de gestion ou d'administration de la sécurité aux plans techniques ou administratifs : conception, production, conditionnement et gestion des réseaux de chiffrement et des éléments secrets.
Auditeur, contrôleur, évaluateur	Parmi les fonctions, citons : la conformité à des réglementations, vulnérabilités découvertes après un audit
Post-auditeur	Architecte de sécurité avec une forte expérience. Il établit la cartographie et oriente les recherches des équipes. Il intervient après une intrusion, un audit.
Opérateur	Applique la politique de sécurité
Intégrateur	Analyse et prend en charge les mécanismes / fonctions de sécurité. Déploie les plateformes et outils.
Développeur (de sécurité)	Assure le sous-ensemble des activités d'ingénierie nécessaires à la réalisation d'éléments, de produits, de logiciels répondant à des exigences de sécurité, en cohérence avec les objectifs qui leur sont alloués et une définition d'architecture d'ensemble
Expert des tests d'intrusion	Aussi appelé hacker éthique. Il pénètre les systèmes pour identifier les failles et les chemins possibles. Il connaît aussi les contre-mesures.

Source : http://www.ssi.gouv.fr/particulier/formation/profils-metiers-de-la-cybersecurite/ *La rédaction*

Xamarin Test Cloud : ne subissez plus la fragmentation (sur mobiles)

L'écosystème des mobiles connaît une évolution rapide et son paysage est très hétérogène. Beaucoup plus que ce que l'on a connu pendant l'âge d'or des PC. Entre 2009 et 2014, les systèmes Android sont passés d'un taux d'adoption de 4% à plus de 76%. Pas loin de 20 000 périphériques utilisent Android, leader mondial des OS mobiles, et tous ces périphériques ont à la fois un « form factor » différent et une personnalisation de la part des constructeurs. En parallèle, Apple supporte 24 configurations différentes d'iPhone et d'iPad ; en ne citant que l'iPhone 6, il y a déjà les déclinaisons iPhone 6, 6 Plus, 6s et 6s Plus.

Nicolas Humann
Exakis

Même si une entreprise s'engage à supporter une plateforme unique, il reste cependant très difficile de garantir la qualité et la cohérence de l'application sur tous les « form factors ».

L'approche qualité sur les applications mobiles

Les applications des grands éditeurs comme Instagram, Dropbox ou Facebook Messenger ont façonné une forte attente des utilisateurs pour toutes les applications ; elles doivent avoir un design moderne, être ergonomiques, rapides et être mises à jour régulièrement.

En moyenne, un utilisateur, ne consacre que quelques secondes à une application. Les 4 premières secondes d'utilisation qui suivent l'installation sont cruciales. Un plantage ou un message mal affiché dégradent rapidement l'impression générale. Une mauvaise note sur les stores est très vite arrivée. Les applications employées en entreprises exigent une expérience utilisateur similaire. Ils abandonnent tout aussi rapidement une application « moche », lente et qui réduit la productivité alors qu'elle est sensée l'améliorer. Toute une stratégie mobile peut être détruite en quelques minutes... La grande variété des périphériques sur le marché, les versions d'OS, les différentes configurations des constructeurs et operateurs, appelée fragmentation, engendre une très grande difficulté pour les équipes de développement de garantir une qualité optimale.

Les tests sur simulateur

Tester une application sur un simulateur permet d'exécuter l'application dans un environnement utilisé par les développeurs. Cette approche est pratique, réduit les coûts et la mise en œuvre, car elle ne nécessite pas d'achat de matériel.

Néanmoins, les inconvénients sont :

- CPU, mémoire et les performances ne sont pas représentatifs ;
- Le Wifi, GPS et capteurs sont simulés ;
- Les spécificités des OEM et des opérateurs ne sont pas présentes.

Les tests manuels

Beaucoup d'équipes de développement s'appuient sur les tests manuels, utilisant plusieurs périphériques physiques. Généralement les tests s'appuient sur les nouvelles fonctionnalités et ne prennent pas ou peu en compte les tests de non-régression.

Les tests manuels sont généralement attribués à une équipe QA dédiée. Elles estiment généralement que les utilisateurs finaux ont les mêmes smartphones, alors que généralement, l'application est téléchargée sur des centaines de périphériques différents.

Néanmoins, même les tests manuels les plus rigoureux ont des limites, telles que :

- Faible couverture des familles de smartphones ;
- Résultat des campagnes lentes, se comptant en semaines ;
- Faille humaine, manque de rigueur.

Les tests humains représentent un investissement important. Il faut compter les moyens humain, l'achat et la gestion des smartphones, en évitant les pertes (ou vols), et la gestion des mises à jour (surtout quand elles sont automatiques). Tout cela réduit l'agilité et l'innovation des équipes de développement.

L'approche Xamarin Test Cloud

En contraste des tests manuels ou sur simulateurs, Xamarin Test Cloud permet aux équipes de tester l'ensemble des fonctionnalités sur plus d'un millier de smartphones différents et ce à chaque nouvelle version de l'application. Xamarin Test Cloud est une solution Cloud qui permet d'automatiser les tests UI sur un maximum de smartphones/tablettes avec un minimum d'effort. Trouver les bugs avant la publication réduit les cycles de développement, de maintenance et diminue le time to market pour les nouvelles fonctionnalités. A ce jour, uniquement les plateformes iOS (iPhone et iPad) et Android sont supportées. Il n'est pas encore possible d'effectuer ces tests sur les plateformes Windows.

L'automation des tests

Xamarin Test Cloud permet le développement des tests d'interface simulant l'interaction d'un utilisateur grâce au support des Framework UI Test et Calabash.

- Xamarin.UITest est un Framework qui permet aux tests d'être écrits en C# en utilisant la bibliothèque de tests NUnit. Ce Framework est bien connu des développeurs ayant l'habitude de développer des tests unitaires.
- Calabash est un Framework qui permet aux tests d'être écrits en Ruby utilisant Cucumber. Calabash est bien adapté au « Behavior Driven Development » , une méthodologie qui met l'accent sur la création des « *spécifications exécutables* ». Ce sont des tests qui peuvent être écrits dans une langue business par tous ceux qui suivent les règles grammaticales imposées par Cucumber.

Quel que soit le Framework utilisé pour développer, ces tests automatisés vont exécuter l'interface utilisateur de l'application et valider la conformité de l'application.

Une fois les tests créés et validés sur le poste de développement, l'application mobile et les tests sont téléchargés vers Xamarin Test Cloud, qui va l'installer et exécuter les tests sur des centaines de périphériques physiques. Une fois l'exécution terminée, Xamarin Test Cloud envoie une notification avec les résultats complets des tests. Fig.1.

Que vous utilisiez Xamarin.UITest ou Calabash, l'application et les tests

sont soumis à Xamarin Test Cloud en ligne de commande, un processus souvent désigné comme un « Test run ». La soumission peut être effectuée manuellement ou dans le cadre d'une intégration continue qui soumet automatiquement la nouvelle application et les tests. Fig.2.

Démarrer avec Xamarin.UITTest

Xamarin.UITest permet de développer des tests UI avec NUnit pour des applications iOS et Android. Il s'intègre étroitement avec les projets Xamarin.iOS et Xamarin.Android mais ils peuvent également être utilisés avec les projets iOS et Android écrits nativement en Objective-C et Java. Xamarin.UITest est la bibliothèque d'automatisation qui permet aux tests NUnit de s'exécuter sur des appareils Android et iOS. Les tests interagissent avec l'interface utilisateur tout comme le ferait un utilisateur : saisir du texte, toucher les boutons, et en effectuant des gestes, comme par exemple les swipes.

Vous pouvez développer vos tests sur Visual Studio ou Xamarin Studio. Il existe des modèles de projets permettant d'initialiser l'ensemble des prérequis nécessaires. Typiquement, un UITest est écrit comme une méthode, ce qui correspond à un test. La classe contenant la méthode est appelée un « test fixture ». Elle contient un ensemble de tests logiques. Elle est responsable de l'initialisation et du nettoyage une fois l'exécution terminée. Chaque test doit respecter le pattern « Arrange-Act-Assert ».

- Arrange : le test initialise tout ce dont il a besoin ;
- Act : le test interagit avec l'application, entre du texte, clique sur des boutons ;
- Assert : le test examine le résultat du test et affiche un message en fonction.

Toutes les interactions avec l'application mobile sont effectuées avec une instance de Xamarin.UITest.IApp. Cette interface définit toute les méthodes permettant d'initialiser votre application avec l'agent de test et d'interagir avec. Il y a aujourd'hui 2 implémentations de cette interface :

- Xamarin.UITest.iOS.iOSApp. Cette classe permet les tests sur iPhone et iPad ;
- Xamarin.UITest.Android.AndroidApp. Cette classe permet les tests sur les systèmes Android.

Il est recommandé d'instancier des objets iOSApp et AndroidApp dans chaque nouvelle classe de tests. Pour cela, il faut utiliser les helpers ConfigureApp et choisir la plateforme concernée.

Android

```
[TestFixture]
public class Tests
{
  AndroidApp app;

  [SetUp]
  public void BeforeEachTest ()
  {
    app = ConfigureApp
      .Android
      .ApkFile ("../../../Droid/bin/Debug/myapp.apk")
      .StartApp ();
  }
}
```

iOS

```
[TestFixture]
public class Tests
{
  iOSApp app;

  [SetUp]
  public void BeforeEachTest ()
  {
    app = ConfigureApp
      .iOS
      .AppBundle ("/../iOSAppProject/bin/iPhoneSimulator/Debug/iosapp.app")
      .StartApp ();
  }
}
```

Interagir avec l'interface utilisateur

Les méthodes

Les classes iOSApp et AndroidApp contiennent plusieurs méthodes permettant d'interagir avec le périphérique et de simuler des actions, nous pouvons citer les plus courantes :

- Tap ;
- DoubleTap ;
- EnterText ;
- SwipeLeft, SwipeRight ;
- ScrollTo, ScrollUp, ScrollDown.

Ces méthodes prennent en paramètre un délégué Func<AppQuery, AppQuery> qui permet de localiser un élément précisément dans l'arborescence des composants graphiques. En effet, Xamarin Test Cloud ne prend pas en compte des actions sur des coordonnées X-Y, mais directement sur des contrôles graphiques. Par exemple, pour un bouton ayant pour ID SaveButton :

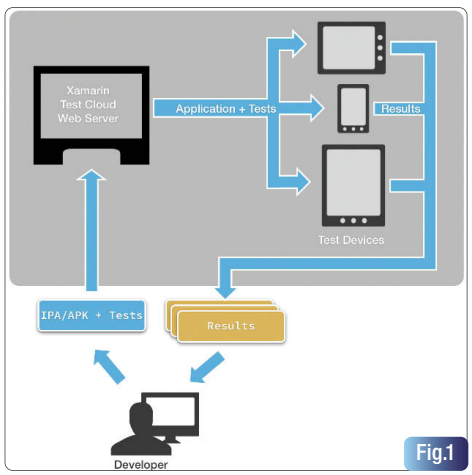


Fig.1

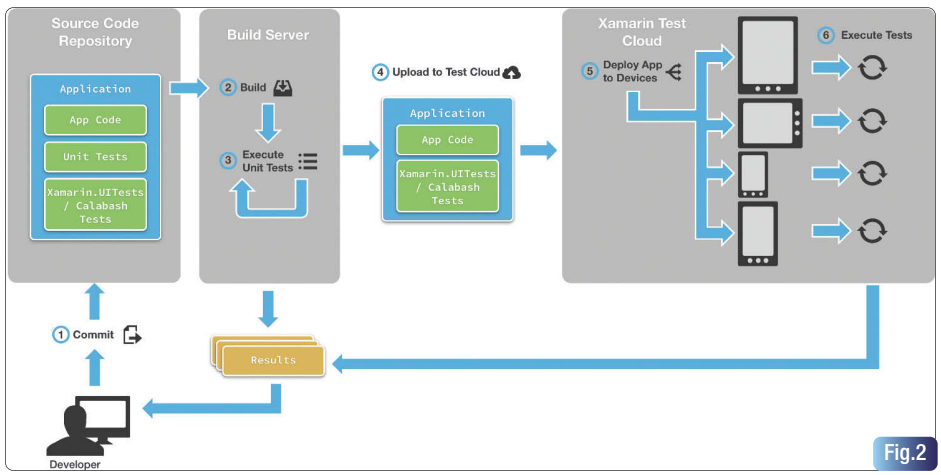


Fig.2

```
[TestFixture]
public class Tests
{
  AndroidApp app;

  [SetUp]
  public void BeforeEachTest ()
  {
    app = ConfigureApp
      .Android
      .ApkFile ("../../../Droid/bin/Debug/myapp.apk")
      .StartApp ();
  }

  [Test]
  public void SaveTheForm ()
  {
    app.Tap(c=>c.Marked("SaveButton"));
  }
}
```

Pour suivre précisément les étapes de test et le rendu graphique, vous pouvez activer la prise de capture d'écran.

```
[TestFixture]
public class Tests
{
  AndroidApp app;

  [SetUp]
  public void BeforeEachTest ()
  {
    app = ConfigureApp
      .Android
      .ApkFile ("../../../Droid/bin/Debug/myapp.apk")
      .EnableLocalScreenshots()
      .StartApp ();
  }

  [Test]
  public void SaveTheForm ()
  {
    app.Tap(c=>c.Marked("SaveButton"));
    app.Screenshot("Click sur le bouton save");
  }
}
```

Le chargement d'un écran

La plupart des écrans qui accèdent à des données distantes affichent un écran de chargement. Pour simuler le comportement utilisateur et garantir que le chargement soit terminé, les API fournissent la méthode WaitForElement. Il est ainsi possible de mettre en pause l'exécution du test et de positionner un timeout.

```
app.WaitForElement (c => c.Marked ("SaveButton"));
```

Localiser les éléments

Malheureusement tous les éléments que l'on souhaite atteindre n'ont pas forcément d'identifiant unique. Pour accélérer le développement et la validation des interactions des tests, il existe un utilitaire très important Repl. Pour pouvoir lancer Repl, il suffit de l'appeler dans un test vide et d'activer le debug sur ce test. Le test se fige, et l'utilitaire Repl s'exécute.

```
  [Test]
  public void NextTest ()
  {
    app.Repl();
  }
```

Repl est un utilitaire en ligne de commande qui permet d'interagir avec l'application en temps réel. Il permet de lister l'arborescence des éléments et de lancer des actions.

```
App has been initialized to the 'app' variable.
Exit REPL with ctrl-c or see help for more commands.

>>>
```

Les commandes exécutables dans Repl utilisent la variable app, la même que nous avons créée dans notre code. Repl supporte l'IntelliSense avec la touche Tab, et permet de retrouver les mêmes méthodes citées précédemment. Pour lister l'arborescence de l'écran, il suffit de taper : tree

```
App has been initialized to the 'app' variable.
Exit REPL with ctrl-c or see help for more commands.

>>> tree
[UIWindow > UILayoutContainerView]
 [UINavigationTransitionView > ... > UIView]
  [UITextView] id: "CreditCardTextField"
   [\_UITextContainerView]
  [UIButton] id: "ValidateButton"
   [UIButtonLabel] text: "Validate Credit Card"
  [UILabel] id: "ErrorrMessagesTestField"
 [UINavigationBar] id: "Credit Card Validation"
  [\_UINavigationBarBackground]
   [\_UIBackdropView > \_UIBackdropEffectView]
   [UIImageView]
  [UINavigationItemView]
   [UILabel] text: "Credit Card Validation"
>>>
```

Pour cliquer sur le bouton « ValidateButton », il suffit d'appeler le code :

```
App has been initialized to the 'app' variable.
Exit REPL with ctrl-c or see help for more commands.

>>> app.Tap(c=>c.Marked("ValidateButton")
```

L'action s'effectue en temps réel dans l'application. Pour l'appliquer dans le code de test, il suffit de taper la commande copy et de coller le code dans Visual Studio ou Xamarin Studio !

Xamarin Test Recorder

Comme nous l'avons vu, pour écrire le code des tests d'interface, nous avons la méthode manuelle et l'utilisation de Repl. Il manque en effet un enregistreur d'actions qui génère le code automatiquement. Depuis Aout

2015, Xamarin a mis à disposition en version preview l'outil Test Recorder. Test Recorder est aujourd'hui uniquement disponible sur Mac. Il est néanmoins possible d'enregistrer des tests pour iOS et Android, mais uniquement sur Mac. Fig.3.

Les étapes principales d'enregistrement sont :

- Déployer l'application sur un simulateur / émulateur ou sur un périphérique physique ;
- Activer l'enregistrement et manipuler l'application ;
- La fenêtre de log s'enrichit des actions que vous effectuez ;
- Vous pouvez rejouer les actions enregistrées ;
- Exporter le code généré dans un fichier cs.

Exécuter et analyser

Une fois le code des tests écrit et validé, l'exécution sur Xamarin Test Cloud est possible. Disponible sur http://testcloud.xamarin.com, vous pouvez créer votre application et sélectionner les périphériques que vous souhaitez utiliser. Il existe près de 2000 périphériques différents, en termes de marques, modèles et versions d'OS.

Le tableau de bord permet de comparer les différents « Test Run », d'afficher l'évolution de la taille du package, la consommation CPU et mémoire dans le temps. De plus, les résultats sont comparés en fonctions de la version d'OS, du « form factor » et du modèle. Fig.4.

Une capture d'écran correspond à chaque étape de test et à chacun des périphériques. Il est ainsi possible de visualiser rapidement le rendu de l'application et de détecter des anomalies. Les tests qui échouent sont facilement identifiables. Fig.5.

Un clic sur une capture d'écran permet d'accéder à un reporting détaillé. Il propose :

- Une capture d'écran en haute résolution ;
- La consommation CPU et mémoire ;
- Les logs du test ;
- Les logs systèmes du périphérique. Fig.6.

Conclusion

Xamarin Test Cloud est une véritable plateforme permettant d'industrialiser les tests des applications mobiles et de garantir une qualité optimale pour vos utilisateurs. Elle fluidifie les cycles de développement et s'adapte aux méthodologies Agiles. La fragmentation des périphériques n'est plus subie, elle est maintenant maitrisée.

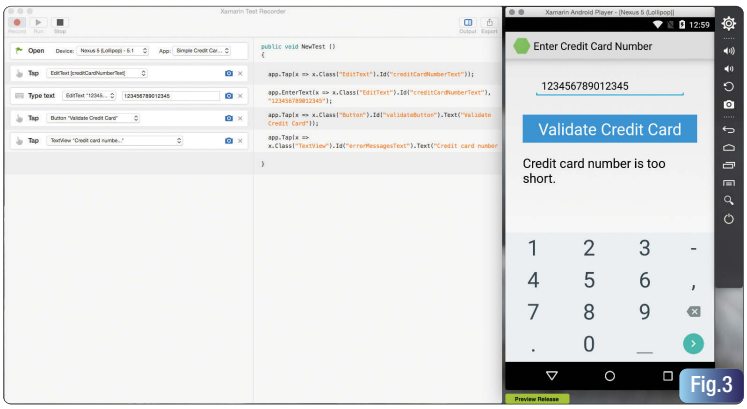


Fig.3

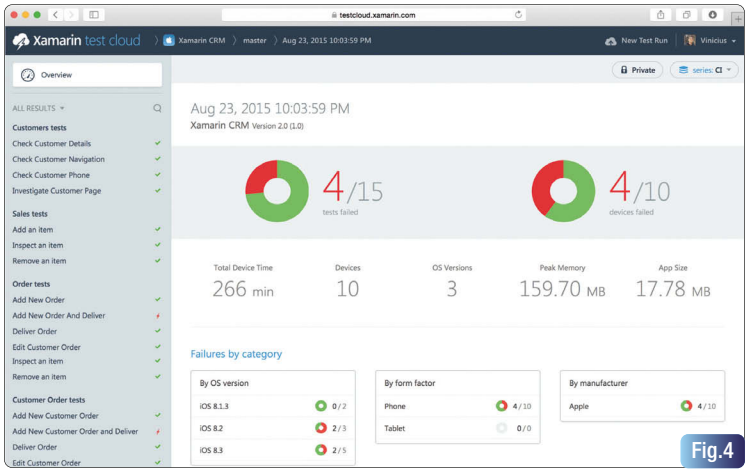


Fig.4

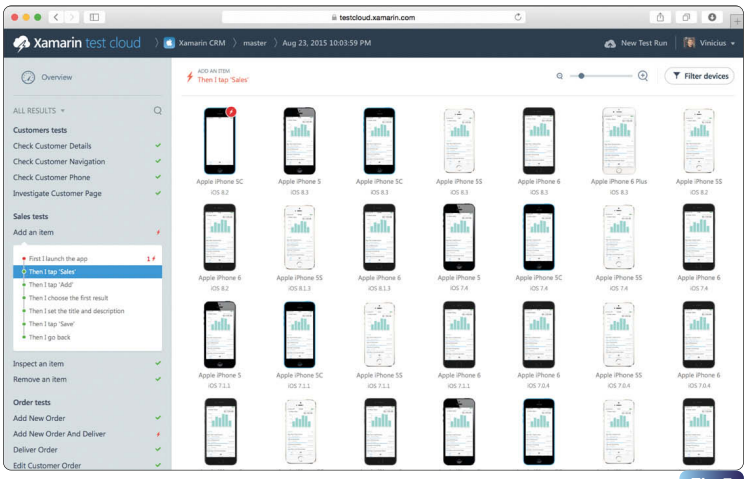


Fig.5

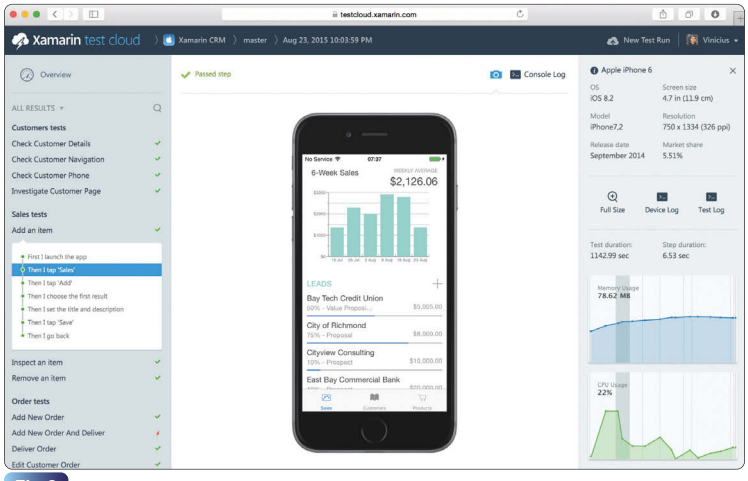


Fig.6

À lire dans le prochain numéro n°194 en kiosque le 27 février 2016

Mon code est-il légal ?

Trouver l'origine des codes sources réutilisées, connaitre les licences des codes, les bons réflexes

Le code est un art

Faire un code propre et bien documenter : oui c'est possible !

Maker

Construire une borne d'arcade avec une Raspberry Pi

Bonnes pratiques pour écrire du code Java 8 sécurisé

Les challenges auxquels les développeurs Java doivent faire face sont nombreux. Parmi ceux-ci, garantir la sécurité d'un programme est sûrement l'un des plus ardus. Fort heureusement, la plateforme est bâtie sur une architecture sécurisée favorisant la bonne exécution des programmes Java. Néanmoins, un certain nombre de pratiques doivent être respectées lors de l'écriture de codes Java afin de ne pas créer de failles potentielles. Cet article propose un passage en revue des bonnes pratiques à connaître pour écrire du code Java 8 sécurisé.

Sylvain SAUREL
Ingénieur d'Etudes Java / Android
sylvain.saurel@gmail.com – www.all4android.net

En écrivant des programmes Java, trop peu de développeurs ont en tête les bonnes pratiques permettant de garantir un code sécurisé à l'exécution. Alors qu'une partie du travail est déjà réalisée par la plateforme Java via son architecture, l'autre partie du travail doit être faite au niveau du code. En effet, alors que la plateforme peut protéger les utilisateurs et systèmes de programmes hostiles, elle ne peut plus rien contre des erreurs d'implémentations sur du code sensé être sûr. De telles erreurs conduisent à la création de failles pouvant impacter l'accès à des fichiers, des imprimantes, des webcams, des microphones ou le réseau. Tout ceci pouvant provoquer des pertes de données confidentielles ou encore de l'espionnage de machines.

La plateforme Java réduit les erreurs de programmation et limite les surfaces d'attaque en proposant un langage type-safe s'exécutant au sein d'une machine virtuelle proposant une gestion automatique de la mémoire et des dépassements sur les tableaux. Les nombreuses bibliothèques tierces disponibles se chargent également de réaliser des contrôles de sécurité poussés ce qui amène les programmes Java à être plus résistants aux dépassements de piles ou à des attaques par buffer overflow que des programmes écrits en C ou C++.

Afin de bénéficier au mieux de ces bases solides en matière de sécurité, le développeur doit s'astreindre à respecter au sein de son code différentes pratiques que nous allons détailler par la suite. Ces dernières s'appliquant sur tous types d'applications Java, du desktop aux applets en passant par les composants et les bibliothèques tierces. Elles vont en effet de la gestion des informations confidentielles à la validation des entrées en passant par la résistance aux attaques par déni de services.

Fondamentaux

Un certain nombre de pratiques sont communes à tous les langages. Parmi celles-ci, il est toujours préférable d'écrire un code simple afin de faciliter sa lisibilité ce qui rendra plus aisée la découverte de failles éventuelles. Dès la phase de conception, la sécurité doit être intégrée à la réflexion car il est plus difficile de sécuriser un système déjà créé. Il convient ainsi d'encapsuler correctement les membres d'une classe en restreignant son accessibilité au maximum et en limitant son extensibilité s'il n'est pas prévu qu'elle soit étendue. Augmentant considérablement la dette technique, le code dupliqué est également un casse-tête en termes de sécurité puisqu'il est difficile de propager des modifications sur chaque copie.

En dépit des efforts des développeurs, un code ne peut être exempt de défauts une fois les revues de code terminées. Pour se prémunir de failles oubliées, il peut être intéressant d'exécuter un code en réduisant ses privilèges. Dans ce domaine, la plateforme Java fournit un mécanisme de sécurité performant pouvant être implémenté statiquement en réduisant les permissions d'exécution via des fichiers définissant la politique de sécurité ou dynamiquement au sein du code en utilisant la méthode `doPrivileged` de la classe `java.security.AccessController`. Ces mécanismes sont également accessibles aux applications RIA via les paramètres de lancement d'une applet ou au sein du fichier JNLP.

Java étant un langage avant tout orienté objet, le recours à la classe `SecurityManager` pour vérifier les permissions dynamiquement doit être fait en dernier ressort. Une bonne pratique consistant à définir des points de contrôle clairement identifiés et retourner les objets autorisés qui devront être retenus dans le code appelant afin de limiter les contrôles de permissions qui restent coûteux.

Déni de services

Les attaques par déni de services, plus connues sous le nom d'attaques DoS, sont un grand classique. Il faut donc redoubler de prudence avec des bouts de code pouvant faire un usage disproportionné des ressources systèmes. Parmi les exemples d'attaques, on peut citer :

- Le chargement d'une image large au format SVG ;
- Un graphe d'objets construit via du texte ou un flux binaire dont l'espace occupé en mémoire peut poser problème ;
- Les attaques type "Zip bombes" où seul l'espace mémoire des objets compressés est borné sans positionner de limites sur l'espace occupé une fois les objets décompressés ;
- La sérialisation / désérialisation de beans Java au format XML doit être réalisée avec précaution car les contenus XML peuvent être malicieux et provoquer des dépassements en mémoire. La taille du flux traité peut être limitée en renseignant la valeur XMLConstants.FEATURE_SECURE_PROCESSING ;
- Prendre garde lorsque des itérations sont faites sur des données d'entrée à ce que chaque itération fasse progresser le travail afin d'éviter des boucles infinies entraînant un dépassement de pile.

Les fichiers, les locks ou la mémoire allouée manuellement nécessitent une libération explicite au niveau du code Java à la fin de leur utilisation. Même

les programmeurs les plus chevronnés peuvent quelquefois oublier de réaliser ces libérations. En prenant l'exemple d'un fichier pour des opérations d'entrées / sorties, une bonne pratique consiste à appliquer le pattern "Execute Around Method" comme suit :

```
public R readFileBuffered(InputStreamHandler handler) throws IOException {
 try (final InputStream in = Files.newInputStream(path)) {
  handler.handle(new BufferedInputStream(in));
 }
}
```

On remarque ici l'utilisation de la syntaxe try-with-resource introduite dans Java 7. Combinée à une lambda expression de Java 8, on peut obtenir un code concis et lisible pour faire la somme des données d'un flux d'entrée :

```
long sum = readFileBuffered(InputStream in -> {
 long current = 0;
 for (;;) {
  int b = in.read();
  if (b == -1) {
   return current;
  }
  current += b;
 }
});
```

Pour les ressources ne bénéficiant pas du support de la syntaxe try-with-resource, il faut continuer à utiliser le processus d'acquisition et de libération classique basé sur try-finally.
Par défaut, le langage Java réalise un grand nombre de vérifications sur les tailles limites de tableaux ce qui limite les attaques potentielles par ce biais. Cependant, certaines opérations sur les types primitifs peuvent provoquer des overflows plus pernicieux. Ceci est problématique sur des ressources persistantes, telles que l'espace disque, pour lesquelles un reboot ne suffira pas à endiguer le problème. Pour vérifier une limite de taille avec des grandes valeurs, plutôt que de tester l'expression current + max qui pourrait provoquer un overflow sur une valeur négative, il faut procéder de la sorte :

```
private void checkGrowBy(long extra) {
 if (extra < 0 || current > max - extra) {
  throw new IllegalArgumentException();
 }
}
```

Informations confidentielles

Les informations confidentielles manipulées par une application doivent être lues uniquement dans un contexte précis et limité. En outre, les données considérées comme fiables ne doivent pas être exposées afin de limiter les risques de falsification. Au niveau du code, il faut absolument penser à purger toutes les informations sensibles des exceptions remontées. Par exemple, si une méthode appelle un constructeur d'un objet java.io.FileInputStream afin de lire un fichier mais que celui-ci est absent, une exception java.io.FileNotFoundException contenant le chemin vers le fichier est lancée mais ne doit pas être propagée à l'appelant sous peine d'exposer des informations sur le système de fichiers. Enfin, trop souvent encore, on voit apparaître sur des pages HTML des stack traces d'exceptions donnant des informations à de potentiels hackers. Il faut désactiver l'affichage des stack traces sur ces pages.
Au niveau des logs, il ne faut jamais y faire figurer d'informations confidentielles sur des utilisateurs telles qu'un mot de passe ou un numéro de sécurité sociale. Les propriétés de classes contenant ce type d'informations doivent être marquées transient pour ne pas être sérialisées et donc stockées dans des fichiers. Enfin, une pratique plus pointue peut être de purger directement la mémoire une fois l'utilisation d'une donnée sensible réalisée sans attendre le passage du garbage collector. Cette pratique extrême permet de se prémunir de copies éventuelles faites par la machine virtuelle sur le disque par exemple.

Injection et Inclusion

Tout aussi classique, l'attaque par injection ou inclusion de données malicieuses vise à modifier le comportement attendu d'un programme. Les données injectées sont bien souvent au format texte et cherchent à profiter d'une mauvaise gestion des caractères spéciaux ou des caractères non échappés. La première bonne pratique consiste à toujours valider le format des données d'entrée en s'appuyant sur des bibliothèques de code tierces éprouvées.
Pour un programme manipulant du SQL, il est préférable d'éviter une utilisation dynamique, encore plus si des données non fiables sont employées pour compléter des requêtes. Les attaques par injection SQL prennent souvent la forme d'une entrée au sein de laquelle un caractère simple quote précède du SQL. Au niveau Java, la bonne pratique est d'utiliser l'API JDBC et ses classes java.sql.PreparedStatement ou java.sql.CallableStatement en évitant java.sql.Statement. En effet, ces deux classes de plus haut niveau prennent soin de faire les vérifications pour éviter les injections SQL. Voici un exemple de requête SQL sécurisée avec PreparedStatement :

```
String sql = "SELECT * FROM User WHERE id = ?";
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1, id);
ResultSet rs = stmt.executeQuery();
```

Si un programme manipule du XML ou du HTML, utiliser prioritairement des bibliothèques tierces éprouvées afin de limiter les risques d'attaques par injection XML ou les vulnérabilités de type XSS (Cross-Site Scripting). Toujours côté XML, il faut restreindre l'inclusion de documents aux pages ayant pour origine le même site Web. Plus vraiment usité, le format BMP est à éviter car il peut référencer des fichiers ICC dont le contenu n'est pas forcément fiable.
Malgré l'émergence de JavaFX, l'API Swing demeure très répandue sur les applications desktops. De nombreux composants de cette API peuvent interpréter un contenu texte commençant par la balise <html> comme du HTML. Si ce contenu provient d'une source inconnue, il est préférable de désactiver le rendu HTML en positionnant au niveau du composant même la propriété html.disable à true :

```
component.putClientProperty("html.disable", true);
```

Toujours au rayon injection et inclusion, il faut prendre garde aux attaques liées aux nombres flottants. En effet, lors de l'import de ces nombres, il est important de rester vigilant sur les limites de confiance telles que la valeur NaN (Not a Number) ou Infinite pouvant être injectées via des entrées. Par exemple, en convertissant une donnée d'entrée au format texte via la méthode Double.valueOf . Malheureusement, aucune détection immédiate n'est proposée ce qui implique que le développeur doit y prendre garde en employant le pattern suivant :

```
if (Double.isNaN(input_double_value)) {
 // action pour NaN
```

```
}

if (Double.isInfinite(input_double_value)) {
 // action pour Nombre Infini
}

// traitement classique ...
```

Accessibilité et Extensibilité

Sécuriser un système est une tâche complexe qui peut être simplifiée en réduisant au maximum les surfaces d'attaques au sein du code. Un package Java regroupe en son sein un ensemble de classes et d'interfaces. Il ne faut déclarer publiques que les classes et interfaces destinées à être exposées via une API par exemple. Autrement, elles doivent être déclarées en package privé. En suivant le même principe, les membres et constructeurs doivent être déclarés publiques ou protégés uniquement s'ils font partie de l'API. A contrario, il faut les déclarer en package privé pour ne pas exposer leur implémentation.

Les classes et méthodes n'ayant pas vocation à être dérivées doivent être déclarées final. En effet, les classes et méthodes qui ne sont pas marquées comme telles peuvent être malicieusement surchargées par un attaquant. Ainsi, il est plus facile de vérifier la sécurité d'une classe n'autorisant pas l'héritage. La bonne pratique consiste à privilégier la composition à l'héritage comme on peut le voir avec la classe suivante :

```
public final class MyClass {
 private final Behavior behavior;

 private MyClass(Behaviour behavior) {
  this.behavior = behavior;
 }

 public static MyClass newMyClass(Behavior behavior) {
  // validation des arguments
  // vérifications sécurité
  return new MyClass(behavior);
 }
}
```

Dans tous les cas, il est évident que bien connaître le modèle objet de Java est un atout en vue de sécuriser son code. L'accès aux instances de ClassLoader autorise certaines opérations pouvant être indésirables comme :

- L'accès à des classes auxquelles le code client n'aurait pas eu accès ;
- Retrouver des informations sur les URLs des ressources ;
- L'activation ou non des assertions.

Il est donc bon de limiter l'exposition des instances de ClassLoader autant que faire se peut.

Validation des entrées

Les entrées utilisateurs et celles des méthodes doivent toujours être validées. Si la plateforme effectue certaines vérifications par défaut sur les types, les limites des tableaux, ou l'utilisation de bibliothèques, le code natif lui ne propose pas ce type de vérifications. De fait, dès lors qu'un programme Java fait appel à des méthodes natives, il faudra les encapsuler via des méthodes wrappers pour valider les paramètres d'entrée :

```
public final class NativeMethodWrapper {
 private native void nativeOperation(byte[] data, int offset, int len);

 public void doOperation(byte[] data, int offset, int len) {
  // copie des données
  data = data.clone();
  // validation des entrées ...
  if (offset < 0 || len < 0 || offset > data.length - len) {
   throw new IllegalArgumentException();
  }

  nativeOperation(data, offset, len);
 }
}
```

Mutabilité

Inoffensive à première vue, la mutabilité peut causer des problèmes de sécurité. Il est donc important de rendre les types valeurs immuables en marquant final les classes n'ayant pas vocation à être dérivées. En outre, cacher les constructeurs offre plus de flexibilité lors de la création d'instances en termes de validation ou de mise en cache. Le design pattern Builder est une solution à suivre. Lorsqu'une méthode de classe renvoie un objet interne mutable, il faut en créer une copie pour éviter que le code appelant puisse le modifier :

```
public class MyClass {
 private final java.util.Date date;
 // ...
 public java.util.Date getDate() {
  return (java.util.Date) date.clone();
 }
```

Le principe est le même pour les entrées mutables d'une classe qui doivent être copiées via un constructeur de copie par exemple. En respectant ces règles, il devient vital lors de la conception de classes valeurs mutables de proposer des constructeurs de copie ou des méthodes de copie publiques. Si l'état interne d'une classe doit être accessible et modifiable de manière publique, il faut le déclarer privé et fournir des méthodes wrappers publiques permettant de valider d'éventuelles modifications de l'état :

```
public final class WrappedState {
 private String state;

 public String getState() {
  return state;
 }

 public void setState(final String newState) {
  this.state = processValidation(newState);
 }

 public static String processValidation(final String state) {
  if (...) {
   throw new IllegalArgumentException();
  }

  return state;
 }
}
```

Les données publiques et statiques doivent être final puisqu'un éventuel appelant peut modifier leur contenu. En sus, il faut être prudent avec les

objets de ce type qui ne sont pas immutables. C'est le cas des collections par exemple qu'il faudra prendre soin de rendre immutables avant une exposition publique :

```
import static java.util.Arrays.asList;
import static java.util.Collections.unmodifiableList;

public static final List<String> names = unmodifiableList(asList("Sylvain", "Rachida",
"Adam", "Nathan"));
```

Construction d'objets

Lors de sa construction, un objet est dans une phase critique, car existant, mais pas encore prêt à être utilisé, ce qui représente certaines difficultés à gérer. La première pratique de sécurité à mettre en place est de cacher le constructeur des classes sensibles et de fournir une méthode factory statique à la place pour mieux contrôler leur création. En outre, les constructeurs implicites via désérialisation ou clonage sont à bannir. Pour bloquer la création d'objets restreints via un ClassLoader spécifique, il est bon de renforcer la sécurité en plaçant des points de contrôle via le SecurityManager. Ceci est encore plus vrai au moment de la création d'objets par désérialisation via les méthodes readObject ou readObjectNoData ou encore de la méthode clone d'une classe clonable.

Lancer une exception dans le constructeur d'une classe non finale peut permettre à un attaquant d'obtenir l'accès à une instance partiellement initialisée de la classe. Il faut donc s'assurer qu'une classe non finale soit totalement inutilisable jusqu'à ce que son constructeur soit pleinement exécuté avec succès. Dernier point crucial à respecter, éviter autant que possible d'appeler au sein d'un constructeur des méthodes qui peuvent être surchargées. En effet, avec un tel appel, le développeur donne une référence à l'objet en cours de construction alors que celui-ci n'est pas complètement initialisé.

Sérialisation et Désérialisation

L'API Serialization de Java est une API de haut niveau permettant aux développeurs de déléguer à la plateforme les mécanismes de contrôle des membres d'une classe au moment de la sérialisation.

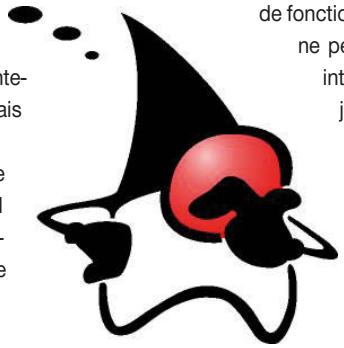
De fait, la prudence doit être plus grande lorsque des opérations de sérialisation et de désérialisation sont réalisées. Dans tous les cas, la désérialisation de données non fiables est à éviter autant que possible.

La règle d'or lorsqu'on associe sérialisation et sécurité consiste à ne jamais rendre sérialisable une classe sensible. En effet, cela reviendrait à créer une interface publique à tous les champs d'une classe et également un constructeur caché mais public.

Au niveau des lambdas, la problématique est identique et les interfaces fonctionnelles de Java 8 ne doivent pas être rendues sérialisables sans avoir considéré ce qu'elles pourraient alors exposer.

Une fois qu'un objet Java a été sérialisé, la plateforme ne peut plus contrôler les accès à ses données et des attaquants potentiels peuvent alors accéder aux données des champs privés d'un objet en analysant le contenu du flux binaire produit. Il faut donc veiller à ne jamais sérialiser les données sensibles d'une classe.

Pour s'assurer de ce point, une des approches est de déclarer les champs sensibles comme transient. Enfin, il ne faut pas hésiter à dupliquer les appels aux vérifications faites par le SecurityManager au sein d'une classe sérialisable à sa construction et dans le readObject :



```
public final class MySensitiveClass implements java.io.Serializable {
 public MySensitiveClass() {
  securityManagerCheck();
  // initialisation …
 }
 // implémenter readObject pour renforcer la sécurité
 private void readObject(java.io.ObjectInputStream in) {
   securityManagerCheck();
   // suite …
 }
}
```

Contrôle des accès

Alors que le langage Java est orienté objet, le mécanisme de contrôle de sécurité de la machine virtuelle est basé sur la pile d'appels. Le modèle de sécurité standard garantit que chaque trame présente dans la pile d'appels a les permissions requises. De plus, les permissions courantes vérifiées correspondent à l'intersection de chacune des permissions des trames liées au contexte en cours de contrôle. Si une trame n'a pas l'une des permissions requises, peu importe sa position dans la pile, le contexte courant n'a de fait pas la permission. L'utilisation de la méthode AccessController.doPrivileged pour restreindre des privilèges à l'exécution est également utile mais réservée à des usages précis.

Un mot sur JNI ...

JNI est l'API standard proposée par la plateforme pour écrire des programmes Java appelant des méthodes natives ou embarquer une JVM au sein d'applications natives. JNI permet d'interagir avec des APIs ne fournissant pas de bindings Java et sert à implémenter des wrappers faisant le lien entre le monde Java et le monde C/C++. A l'exécution, les méthodes natives définies dans une bibliothèque dynamique sont connectées aux méthodes Java marquées par le mot-clé native.

La meilleure mesure de sécurité concernant JNI est d'éviter de recourir à du code natif depuis Java autant que faire se peut. Si l'on ne peut trouver d'alternative, il faut garder à l'esprit les différents types de menaces pesant sur les codes C/C++ puisque la mémoire n'y est pas managée par exemple. Une fois que le code C/C++ appelé via JNI est chargé avec succès, il n'y a plus aucune protection qui peut être assurée par la plateforme Java en termes de visibilité ou de politique de sécurité. De fait, l'utilisation de JNI doit être sécurisée côté Java via des validations sur les données d'entrées / sorties. Enfin, le code Java doit être écrit de façon à prévoir les cas d'échec possibles et à y répondre correctement.

Conclusion

Ecrire un code Java 8 sécurisé est loin d'être une tâche triviale, mais, fort heureusement, la plateforme Java fournit aux développeurs des bases solides pour sécuriser les applications qu'ils développent via un ensemble de fonctionnalités proposées en standard. Néanmoins, la plateforme ne peut prévenir de tout et notamment des éventuelles failles introduites au sein du code. C'est alors aux développeurs de jouer en faisant preuve de pragmatisme et en se posant les bonnes questions pour tirer profit des bonnes pratiques exposées dans cet article et de la puissance de la plateforme afin de réaliser des programmes Java 8 sécurisés et résistants aux attaques. L'approche la plus efficace restant évidemment de ne pas avoir de défauts plutôt que ne pas avoir de défauts évidents.

C# 6.0 : nouveautés et maturité

Le C# est un langage dynamique qui a l'avantage d'évoluer pour le bien du développeur. Celui-ci enlève les fioritures du code et permet au développeur de se concentrer sur l'essentiel, sur les parties importantes du code, l'essence même de celui-ci.

Teddy DESMAS http://blogs.infinitesquare.com/b/tdesmas
Daniel DJORDJEVIC
http://blogs.infinitesquare.com/b/ddjordjevic
Jonathan ANTOINE
http://blogs.infinitesquare.com/b/jonathan
Consultants Infinite Square

A chaque version, le C# apporte de nombreuses fonctionnalités pour faciliter la vie du développeur, sans altérer la qualité et les performances du code de celui-ci, tout en lui permettant d'assurer un minimum sa rétrocompatibilité. Cela fait un long moment que le C# acquiert la capacité d'utiliser un typage dynamique là où il est véritablement utile de l'utiliser; plusieurs fonctionnalités associées aux langages dynamiques et fonctionnels sont arrivées dans C#, menant à un code aussi facile à écrire qu'à maintenir.
La philosophie avec C# 6 reste inchangée : améliorer le quotidien des développeurs ainsi que les scénarios les plus courants auxquels les développeurs font face chaque jour. Et cela, sans ajouter de complexité ni de nouveaux bagages conceptuels trop conséquents. Certaines fonctionnalités sont principalement du sucre syntaxique, d'autres sont des ajouts pour compléter certaines fonctionnalités qui ne semblaient pas finies, ou alors les améliorer dans un sens plus naturel pour le développeur.

Introduction

Pour ceux qui connaissent déjà les nouvelles fonctionnalités de C# 6, des questions peuvent se poser sur l'absence de « killer-feature ». Sans vouloir dénigrer les nouveautés que nous allons présenter ci-dessous, il est important de savoir que les équipes Microsoft étaient focalisées sur l'avancement d'un projet qui est annoncé depuis maintenant longtemps : le nouveau compilateur connu sous le nom de Roslyn ou .NET Compiler Platform. Historiquement, le compilateur C# était une boîte noire. Le développeur écrivait du code et si celui-ci était valide, le compilateur se chargeait de générer une assembly (.exe ou .dll). Cependant, l'implémentation du compilateur était complètement opaque.

Maintenant, Roslyn est open-source, et surtout, il expose des APIs qui exposent elles-mêmes de riches informations vis-à-vis du code source. Nous sommes passés à l'ère du Compiler As A Service (Compilateur en tant que service) :
Les avantages de Roslyn ne sont pas limités à Visual Studio. Des outils open-source comme Omnisharp (permettant de faire du .NET dans n'importe quel éditeur de code), par exemple, sont alimentés par Roslyn, et on peut très bien imaginer que de nombreux outils vont suivre la tendance, ou même que de nouveaux outils vont émerger.
Comme vous pouvez le comprendre, un portage du code des compilateurs actuels pour le code managé semble être une tâche colossale. Toutefois, il sera maintenant plus facile de prototyper et mettre en œuvre de nouvelles fonctionnalités grâce à Roslyn, donc nous allons très probablement faire face à de nombreuses innovations et cela très prochainement.
A savoir, nous avons utilisé Visual Studio Enterprise 2015 pour nos différents tests. Pour ceux qui ne disposent pas d'une licence Visual Studio 2015, Microsoft met à disposition Visual Studio Community 2015, avec laquelle vous pourrez profiter de l'expérience C# 6 sans problème.
Enfin, voyons quelles sont ces nouvelles fonctionnalités !

Static Types en using

Nous sommes tous habitués à la notion de méthodes statiques et à leur utilisation en utilisant le nom de la classe les portant en premier : Console.WriteLine() ;
Avec C# 6, ce n'est plus obligatoire grâce à l'utilisation du mot-clef using static définissant une des classes où le compilateur va aller chercher les méthodes statiques. Par exemple, avant C# 6, nous avions accès aux méthodes Console.ReadLine(), Console.WriteLine() exposées par la classe Console uniquement en utilisant le qualificatif Console, dorénavant, on peut simplement se passer du nom de la classe dans le code.
Avant C# 6 :

```
using System;

namespace CSharpSix
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello world");
    }
  }
}
```

Avec C# 6:

```
using static System.Console;

namespace CSharpSix
{
  class Program
  {
    static void Main(string[] args)
    {
      WriteLine("Hello world");
    }
  }
}
```

Nous voyons ici l'exemple avec System.Console, mais cela marche bien entendu avec des namespaces et classes statiques du développeur aussi :
Avant C# 6 :

```
namespace CSharpSix
{
```

```
  class Program
  {
    static void Main(string[] args)
    {
      PromptHelper.HelloWorld();
    }
  }

  static class PromptHelper
  {
    public static void HelloWorld()
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

Avec C# 6 :

```
using System;
using static CSharpSix.PromptHelper;

namespace CSharpSix
{
  class Program
  {
    static void Main(string[] args)
    {
      HelloWorld();
    }
  }

  static class PromptHelper
  {
    public static void HelloWorld()
    {
      Console.WriteLine("Hello World");
    }
  }
}
```

Cette fonctionnalité permet d'écrire du code beaucoup plus rapidement mais il faut cependant l'utiliser avec parcimonie car il peut vite devenir difficile de trouver l'origine d'une méthode lors de la lecture du code.

String interpolation

Une des features très intéressantes pour le développeur, le « nouveau » String.Format, va vous permettre de formater plus simplement vos chaînes de caractères. Le String.Format peut inciter le développeur à faire des erreurs à cause de la notation {0}, {1} et de la nécessité d'avoir autant de paramètres que d'index spécifiés. Le développeur peut être amené à faire des erreurs à ce niveau-là en inversant des positions ou en oubliant des paramètres. La String interpolation va permettre de mettre directement les variables dans la chaîne de caractères servant de *patron*. Pour cela, on remplace l'index du paramètre par le nom de la variable entre les accolades. Avant C# 6

```
var nom = "Infinite";
var prenom = "Square";
var str = String.Format("Je m'appelle {0} {1}", prenom, nom);
```

Avec C# 6

```
var nom = "Infinite";
var prenom = "Square";
var str = $"Je m'appelle {prenom} {nom}" ;
```

Il est parfaitement possible d'utiliser des expressions, pour, par exemple, inclure d'autres chaînes de caractères. Par exemple, avec le cas classique de la pluralisation d'un terme en fonction de la quantité de celui-ci :

```
int count = 20;
var str = $"Il y a {count} objet{(count > 1 ? "s" : string.Empty)}";
// Affiche "Il y a 20 objets"
```

On met ici l'expression conditionnelle entre parenthèses pour qu'il n'y ait pas de confusion pour le compilateur au niveau du format.
Le développeur peut aussi continuer à utiliser les paramètres d'alignements ou de formatage du texte. Là où, avant C# 6, on spécifiait :

```
var product = "Salade";
var price = 20;
var str = string.Format("Produit : {0, 10}, Prix : {1:D3}€", product, price);
```

Avec C# 6 on spécifie :

```
var str = $"Produit : {product,10}, Prix : {price:D3}€";
```

Initialiseurs avec membres indexés

Depuis C# 3 nous sommes habitués à pouvoir utiliser les initialiseurs de collections de manière simple et intuitive :

```
using System.Collections.Generic;

namespace CSharpSix
{
  class Program
  {
    static void Main(string[] args)
    {
      var products = new List<string>
      {
        "Téléphone",
        "Télévision",
        "Home cinéma"
      };
    }
  }

}
```

Cependant en ce qui concerne un dictionnaire, il faut insérer des paires de clef-valeur, ce qui est moins intuitif, car cette syntaxe ne spécifie pas clairement quelle donnée est la clef et laquelle est la valeur associée à cette clef.

```
using System;
using System.Collections.Generic;

namespace CSharpSix
{
  internal class Program
  {
    private static void Main(string[] args)
    {
```

```
        var zipCodes = new Dictionary<string, string>
        {
          {"67200", "Strasbourg"},
          {"69000", "Lyon"},
          {"75011", "Paris"}
        };
    }

  }
}
```

Vous le savez sûrement déjà, cela est exactement la même chose que de créer le dictionnaire puis d'appeler la méthode Add() explicitement pour chaque ajout :

```
var zipCodes = new Dictionary<string,string>();
zipCodes.Add("67200", "Strasbourg");
zipCodes.Add("69000", "Lyon");
zipCodes.Add("75011", "Paris");
```

L'initialiseur par index permet maintenant, en C# 6, d'initialiser un objet d'une nouvelle manière : en utilisant son indexeur, s'il en possède un. Par exemple, dans notre cas du dictionnaire d'adresses qui possède un indexeur, nous pouvons utiliser la nouvelle syntaxe de cette manière :

```
var zipCodes = new Dictionary<string, string>
{
    ["67200"] = "Strasbourg",
    ["69000"] = "Lyon",
    ["75011"] = "Paris"
};
```

Cette utilisation de l'initialiseur avec indexeur permet de spécifier une valeur pour chaque clef. A plusieurs niveaux, cela est plus élégant, car cette syntaxe différencie concrètement la clef de la valeur.

Il est important de comprendre que ce n'est pas uniquement du sucre syntaxique.

Ce n'est plus la méthode Add() qui est utilisée à ce moment-là (on peut donc utiliser l'initialiseur d'index sans que l'objet ne se doive d'implémenter la méthode Add()), mais bel et bien l'indexeur de l'objet.
Ce qui donnerait plutôt après le passage du compilateur :

```
var zipCodes = new Dictionary<string, string>();
zipCodes["67200"] = "Strasbourg";
zipCodes["69000"] = "Lyon";
zipCodes["75011"] = "Paris";
```

Cela implique notamment, que si une clef existe déjà, sa valeur sera remplacée mais aussi qu'aucune exception ne sera ainsi levée pour cause de clefs dupliquées.

Null Propagation

C'est peut-être LA fonctionnalité qui réjouit énormément de développeurs. La NullReferenceException est un cas relativement simple à gérer, un bon if(monObjet != null) et le tour est joué. Cependant, le bon développeur est fainéant, et surtout, le développeur aime bien produire plus en écrivant le moins possible. Avec la Null Propagation, nous allons pouvoir accéder aux membres d'un objet sans qu'une NullReferenceException soit levée dans le cas où l'objet en question est null.

Avant C# 6 :

```
if (student != null)
{
string name = student.FirstName;
}
```

Avec C# 6

```
string name = student?.FirstName;
```

Ce qu'il faut comprendre, c'est que FirstName ne va être évalué que si student n'est pas null, sinon l'expression entière renvoie null.
On peut bien évidemment aller à plusieurs niveaux, prenons l'exemple avec notre classe Student qui possède une propriété de type Address, cette dernière ayant une propriété City. On évite ainsi de nombreuses conditions imbriquées grâce à cette syntaxe.

```
string city = student?.Address?.City;
```

Il est aussi intéressant de savoir qu'un effet de bord est que lorsque le dernier élément de la chaîne est de type valeur, celui-ci est wrappé dans une version Nullable de ce type :

```
int? zipCode = student?.Address?.ZipCode;
```

Opérateur nameof

Cet opérateur est plutôt simple à comprendre, il envoie tout simplement dans une chaîne de caractères le nom du symbole (méthode, propriété, etc.) qui lui est fourni :

```
// Va renvoyer "WriteLine"
string name = nameof(Console.WriteLine);

// Va renvoyer "FirstName"
string name = nameof(student.FirstName);
```

Le principal atout est de ne plus mettre de nom en dur dans le code pour faire référence à des symboles.
Principalement, lorsqu'on est amené à mettre en place des logs, de la gestion d'erreur, etc., il est important d'avoir des messages explicites qui permettent aux développeurs de pouvoir faire un constat rapide de l'erreur (ou du log). Avec le nameof, le développeur peut, et doit, spécifier un symbole qui existe, sans quoi le code ne compilera pas. Cette contrainte n'existe pas lorsqu'on utilise uniquement des chaînes de caractères, car le compilateur ne fait pas le rapport entre ce que possèdent les chaînes de caractères (le nom du symbole en dur donc) et le nom des différentes variables.
On peut aussi voir un intérêt dans le sens où les outils de refactoring vont automatiquement renommer les références des paramètres fournis aux opérateurs nameof, lorsque le développeur va changer le nom d'une variable par exemple. Oublier de changer le nom d'un paramètre passé sous forme de chaîne de caractères lors d'un renommage de variable était en effet une source d'erreur très courante.

Await dans les blocks catch et finally d'un try

Il y a quelque temps de cela, avec le .Net Framework 4.5 et C# 5.0, Microsoft nous a mis à disposition des techniques simples pour effectuer de l'asynchronisme. Les mots clés async/await ont donc fait leur apparition et sont maintenant connus des développeurs de la sphère .Net. Avec cet apport, nous pouvons réaliser des applications plus réactives et plus performantes sans se soucier ou presque de comment le moteur interprète cela. Pour l'utilisateur final, cela se traduit par une application plus « vivante », n'ayant pas ou peu de phases bloquantes qui la rende inutilisable et

provoque parfois un effet de « freeze ». Beaucoup de développeurs réalisent donc leurs applications en mettant au cœur de leur application ce système d'asynchronisme et essaient de l'utiliser le plus possible au sein de celle-ci.
Certains d'entre vous auront donc essayé de mettre de l'asynchronisme dans un block catch ou finally d'un try, et ce sans succès. Il était alors par exemple nécessaire d'utiliser un booléen pour savoir si une exception était levée et de faire l'attente (await) en dehors du bloc try/catch : ouch.
Microsoft pensait jusqu'alors, qu'il leur était impossible d'implémenter ceci, mais ils ont réussi !
Et nous permettent aisément de réaliser cela, sans avoir besoin de faire des workarounds complexes à lire.
Voici comment utiliser cette nouveauté :

```
public class MyAwesomeClass
{
  public async Task DoSomeAwesomeThingsAsync()
  {
    Console.WriteLine("Début de la méthode");
    try
    {
      Console.WriteLine("Début du travail");
      await DoSomethingAsync();
      Console.WriteLine("Fin du travail");
      throw new Exception();
    }
    catch (Exception e)
    {
      Console.WriteLine("Début du catch");
      await DoSomethingAsync();
      Console.WriteLine("Fin du catch");
    }
    finally
    {
      Console.WriteLine("Début du finally");
      await DoSomethingAsync();
      Console.WriteLine("Fin du finally");
    }
    Console.WriteLine("Fin de la méthode");
  }

  public async Task DoSomethingAsync()
  {
    Console.WriteLine("Début tâche asynchrone");
    await Task.Delay(1000);
    Console.WriteLine("Fin tâche asynchrone");
  }
}
```

Et voici le rendu dans la console :

```
Début de la méthode
Début du travail
Début tâche asynchrone
Fin tâche asynchrone
Fin du travail
Début du catch
Début tâche asynchrone
Fin tâche asynchrone
Fin du catch
Début du finally
Début tâche asynchrone
Fin tâche asynchrone
Fin du finally
Fin de la méthode
```

On voit bien que les tâches asynchrones sont bien exécutées de manière asynchrone et ce même dans les blocks catch et finally. Pour effectuer le même rendu avant l'implémentation de cette nouveauté, nous aurions pu faire ceci pour dans la méthode DoSomeAwesomeThingsAsync :

```
public async Task DoSomeAwesomeThingsAsync()
{
  var exceptionCatched = false;
  Console.WriteLine("Début de la méthode");
  try
  {
    Console.WriteLine("Début du travail");
    await DoSomethingAsync();
    Console.WriteLine("Fin du travail");
    throw new Exception();
  }
  catch (Exception e)
  {
    exceptionCatched = true;
  }

  if (exceptionCatched)
  {
    // For Example log Exception
    Console.WriteLine("Début du catch");
    await DoSomethingAsync();
    Console.WriteLine("Fin du catch");
  }

  Console.WriteLine("Début du finally");
  await DoSomethingAsync();
  Console.WriteLine("Fin du finally");

  Console.WriteLine("Fin de la méthode");
}
```

On voit bien que pour réaliser le même rendu, nous avons dû sortir le traitement asynchrone en dehors du catch et du finally.
Nous avons utilisé l'approche la plus simple en utilisant un booléen que nous contrôlons par la suite, pour savoir si une exception s'est produite et pour effectuer le travail adéquat.
Cependant, cette méthode a ses limites et dans certains cas, sans cet asynchronisme dans les bloc finally/catch, il est difficile d'assurer le bon fonctionnement, le rendu attendu et la maintenabilité du code.

Déclaration des membres d'une classe

Toujours pour améliorer le confort de développeur et réduire le temps de développement, Microsoft ajoute de nouvelles possibilités pour déclarer les membres d'une classe.
Nous allons voir qu'ils privilégient grandement l'initialisation des propriétés au même niveau que leur déclaration plutôt que dans le constructeur.

Initialisation par défaut de propriétés

Dans certains cas, il peut être bon d'affecter une valeur par défaut à une propriété. Pour une propriété avec des accesseurs, nous pouvons maintenant directement définir cette valeur en la spécifiant à la suite de la déclaration de cette propriété. Voici comment se présente cette fonctionnalité :

```
public string FirstName { get; set; } = "Teddy";
public string LastName { get; set; } = "Desmas";
```

Les deux propriétés sont instanciées dès la création de leur classe avant même de passer dans le constructeur de celle-ci. Cependant, on ne peut pas leur assigner de valeur relative au « this ».
Il est impossible d'effectuer le même rendu dans la version précédente de C#. La seule qui pourrait s'en rapprocher, serait d'instancier ces propriétés dans le constructeur ou de ne pas utiliser une propriété automatique en initialisant le champ de stockage.

Propriétés accessibles uniquement en lecture

Il est habituel pour les propriétés devant être accessibles uniquement en lecture, d'avoir un accesseur en lecture publique et un accesseur en écriture privée. Avec C# 6, nous ne sommes même plus obligés de déclarer l'accesseur en écriture.
En effet, si celui-ci n'est pas déclaré, il est automatiquement interprété comme un accesseur existant en readonly (qui peut uniquement être modifié dans le constructeur de la classe). Voici comment on peut déclarer une propriété en C # 6 avec cette nouvelle méthode :

```
public class User
{
  public string DisplayName { get; }
  public string FirstName { get; set; } = "Teddy";
  public string LastName { get; set; } = "Desmas";

  public User()
  {
    DisplayName = FirstName + " " + LastName;
  }
}
```

Nous n'avons pas déclaré d'accesseur en écriture à ma propriété « DisplayName », celle-ci est automatiquement reconnue en tant que readonly et n'est accessible en écriture que dans le constructeur de ma classe. On peut également assigner une valeur par défaut en utilisant la fonctionnalité vue juste au-dessus.
Pour effectuer le même rendu avant l'implémentation de cette fonctionnalité, nous devions déclarer la propriété comme « readonly ».

```
public class User
{

  public readonly string DisplayName;
  public string FirstName { get; set; } = "Teddy";
  public string LastName { get; set; } = "Desmas";

  public User()
  {
    DisplayName = FirstName + " " + LastName;
  }
}
```

Le rendu est donc le même et la différence concernant l'écriture du code n'est pas aussi notable que les autres exemples. En fonction du développeur et/ou des règles de nommage établi, on choisira l'une ou l'autre des solutions proposées.

Ecriture lambda dans les propriétés, fonctions, ...

C'est pour moi, la fonctionnalité qui pourra vous faire gagner le plus de temps d'écriture de code. Elle permet de pouvoir directement écrire le contenu d'une procédure, d'une fonction ou d'une propriété en utilisant la syntaxe lambda et notamment de la fameuse flèche « => ».
Voici ce que l'on peut faire en s'amusant avec ceci :

```
public class User
{
  public readonly string DisplayName;
  public string FirstName { get; set; } = "Teddy";
  public string LastName { get; set; } = "Desmas";
  public int MyNumber => (int) (DisplayName.Length*3.14/2 + 32);

  public User()
  {
    DisplayName = FirstName + " " + LastName;
  }

  public char DetectMostOccuredLetter() => DisplayName.GroupBy(l => l).OrderByDescending
(l => l.Count()).First().Key;

  public void SayMeHello() => Console.WriteLine("Coucou");
}
```

Ici nous avons plusieurs utilisations de cette nouvelle manière d'écrire, avec :

- La propriété « MyNumber » qui est une propriété calculée avec un savant calcul en fonction du « DisplayName ».
- La fonction « DetectMostOccuredLetter » qui détermine en fonction du « DisplayName », la lettre se répétant le plus de fois dans la chaîne.
- La procédure « SayMeHello » qui affiche « Coucou » dans la console.

Voici comment nous aurions dû faire avant :

```
public class User
{
  public readonly string DisplayName;
  public string FirstName { get; set; } = "Teddy";
  public string LastName { get; set; } = "Desmas";
  public readonly int MyNumber;

  public User()
  {
    DisplayName = FirstName + " " + LastName;
    MyNumber = (int)(DisplayName.Length * 3.14 / 2 + 32);
  }

  public char DetectMostOccuredLetter()
  {
    return DisplayName.GroupBy(l => l).OrderByDescending(l => l.Count()).First().Key;
  }

  public void SayMeHello()
  {
```

```
    Console.WriteLine("Coucou");
  }
}
```

On constate donc que le code est plus long, mais dans notre cas, nous obtenons le même résultat. Une différence notable est que dans la version avec les « lambdas », la propriété calculée « MyNumber » est évaluée à chaque lecture. Si cette valeur dépend d'une autre, et que cette dernière est mise à jour, la valeur le sera aussi. Dans le second cas, pour avoir ce même comportement, il faut passer par des fonctions qui vont recalculer la valeur de la propriété à chaque changement de la valeur sous-jacente.

Filtre dans les blocks d'exception

Microsoft a ajouté une fonctionnalité proposée par la CLR et utilisée notamment en F# mais qui n'était pas implémentée jusqu'alors. Grâce à cette fonctionnalité, nous pouvons mettre en place des filtres lorsqu'une exception est levée pour la filtrer plus précisément.
Voici ce que nous pouvons réaliser :

```
try
{
  // Traitement qui peut lever des exceptions
}
catch (Exception e) when (e is NotImplementedException)
{
  // Avertir l'utilisateur qu'une exception a été levée
  // car son traitement n'a pas encore été implémenté
}
catch (Exception e) when (e is AccessViolationException && DateTime.UtcNow.Day == 11)
{
  // Avertir l'utilisateur qu'une exception a été levée
  // car il ne peut pas accéder au traitement le 11 du mois
}
catch (Exception e) when (!LogException(e))
{
  // L'exception n'était pas attendue et n'a pas été loguée
  // Avertir l'utilisateur
}
```

L'exemple de code ci-dessus intercepte les exceptions qui peuvent être levées et effectue en fonction de certains filtres (le type d'exception et la date dans un second temps) des traitements. Dans la dernière condition, l'exception est loguée et si le log ne fonctionne pas, on avertit l'utilisateur.
Pour avoir le même comportement, nous aurions réalisé ceci :

```
Exception exception = null;

try
{
  // Traitement qui peut lever des exceptions
  throw new AccessViolationException();
}
catch (NotImplementedException e)
{
  // Avertir l'utilisateur qu'une exception a été levée
  // car son traitement n'a pas encore été implémenté
}
catch (AccessViolationException e)
{
  if (DateTime.UtcNow.Day == 11)
  {
    // Avertir l'utilisateur qu'une exception a été levée
    // car il ne peut pas accéder au traitement le 11 du mois
  }

  exception = e;
}
finally
{
  if (exception != null)
  {
    if (!LogException(e))
    {
      // L'exception n'était pas attendue et n'a pas été loguée
    }
  }
}
```

On voit donc que le code est plus complexe notamment pour la partie où l'on a deux conditions sur l'exception (le type et la date), on doit stocker l'exception pour effectuer le traitement global si jamais la condition n'est pas validée.

Déroulage de pile

On pourrait croire que les deux exemples de codes ci-dessus sont identiques et que ce n'est que du sucre syntaxique (à ne pas confondre avec le célèbre sucre d'Erstein comme dirait Jonathan Antoine), cependant ce n'est pas le cas !
Lorsqu'on rentre dans un bloc catch, la pile est déroulée et donc les appels de méthodes plus profonds sont aussi dépilés et on perd donc les informations relatives à ceux-ci (appels, contexte, valeur des variables, etc.). La différence vient du fait que sans les filtres d'exceptions, on rentre obligatoirement dans le bloc catch, alors qu'avec les filtres d'exception, si la condition n'est pas valide, la pile ne sera pas déroulée car on ne rentre pas dans le bloc catch.
Cela rend les filtres d'exception très intéressants, non d'un point de vue code ou optimisation des performances de celui-ci, mais d'un point de vue débogage, pour permettre au développeur de garder toutes les informations nécessaires lors de la gestion des erreurs.

Conclusion

En conclusion, les nouveautés qu'apporte C# 6.0 ne vont pas radicalement changer votre code, du moins pas autant que l'ajout de LINQ ou des génériques. Il permet de réaliser un code peut être plus proche de la pensée du développeur, et permet dans certains cas de ne pas avoir à réaliser des algorithmes complexes (notamment à l'async/await).
Certaines de ces nouveautés ne seront pas ou peu utilisées par la plupart des développeurs mais certaines, comme la propagation du null, devraient ravir les développeurs que nous sommes. Pour y avoir goûté, revenir en arrière avec un enchaînement de « if » n'est pas simple et est beaucoup plus lourd à lire.
Pour ceux qui souhaitent en savoir plus sur ce qu'il se passe quand on utilise telle ou telle fonctionnalité et notamment la comparaison entre le code post C#6.0 et le nouveau, nous vous invitons à consulter le site http://tryroslyn.azurewebsites.net/ qui montre comment le compilateur compile le code.

Introduction à Gearman pour le multitâche en PHP

1ère partie

Gearman (anagramme de « Manager ») est un framework applicatif open source conçu pour distribuer des tâches vers plusieurs machines ou processus, en supportant le changement d'échelle. Il permet d'effectuer des traitements en parallèle, de faire de la répartition de charge, et d'appeler des fonctions entre différents langages. En utilisant la redondance et la persistance, il peut être utilisé sans présenter de point individuel de défaillance et en assurant la reprise après erreur.

Guillaume Burlot,
ingénieur développement **Osaxis**

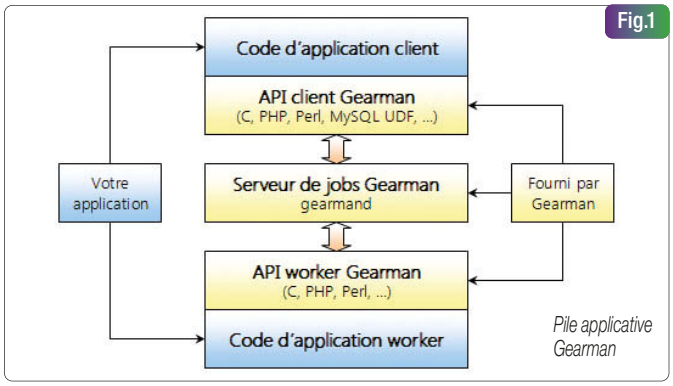
Présentation de Gearman

Gearman utilise trois types d'entités qui communiquent : client, worker, et serveur de jobs :

- Le client envoie des tâches (principalement des demandes d'exécution de job) à un serveur de jobs. Les tâches peuvent être synchrones ou asynchrones, et avoir différents niveaux de priorité (haute, basse, ou normale).
- Le serveur de jobs gère une « file de priorité » de tâches, et transmet chaque job à un worker disponible capable de le traiter.
- Le worker exécute chaque job envoyé par un serveur et envoie une réponse au serveur qui la transmet au client initiateur de la demande.

Un job pourrait correspondre par exemple au redimensionnement d'une image, dans le contexte d'une application Web recevant jusqu'à plusieurs milliers d'images uploadées par seconde.

Gearman fournit des interfaces de programmation (pour différents langages) destinées à être appelées par le code des applications (clients et workers) pour communiquer avec le serveur de jobs Gearman (figure 1, recréée d'après le site officiel gearman.org). Fig.1.



Mise en application initiale : sans Gearman

Nous allons créer une application Web très simple avec le fonctionnement suivant : on peut soumettre des chaînes de caractères (cinq phrases), l'application effectue sur chacune un traitement relativement long (pour l'exemple, cela se résumera à transformer les lettres en capitales, mais en simulant un temps de calcul de trois secondes). L'application enregistre ensuite le résultat dans un fichier (un fichier distinct par phrase soumise), et une page affiche la liste des fichiers existants et leurs contenus. Nous utiliserons deux machines (en réseau local), déjà installées et configurées :

- « www » (adresse IP 192.168.0.100) : serveur Web (HTTP) Apache + PHP.
- « files » (adresse IP 192.168.0.150) : serveur de fichiers FTP.

Le code

Nous utilisons le langage PHP, la version du serveur est 5.5 (le code est compatible 5.4). Nous écrivons les deux pages de notre application Web : liste et formulaire. Pour garder l'exemple concis, nous ne faisons pas de séparation MVC, et nous plaçons nos deux fichiers PHP dans le dossier DocumentRoot d'Apache sur le serveur www. Nous factorisons juste quelques parties communes dans trois fichiers à inclure (dans un sous-dossier « inc »). Voici le code des cinq fichiers : list.php :

```
<?php

require_once __DIR__ . '/inc/config.php'; // for FOO_FILES_DIR

$results = array();
foreach (scandir(FOO_FILES_DIR) as $filename) {
  if ($filename[0] !== '.') {
    $filePath = FOO_FILES_DIR . '/' . $filename;
    $results[$filename] = array(
      'created_at' => filemtime($filePath),
      'result' => json_decode(file_get_contents($filePath), true),
    );
  }
}

$tpl_title = 'Liste';
require __DIR__ . '/inc/html_begin.php'; // uses $tpl_title, defines e()
?>

<h1>Liste des fichiers créés</h1>

<table border="1">
  <tr>
    <th>Fichier</th>
    <th>Date de création</th>
    <th>Phrase originale</th>
    <th>Phrase transformée</th>
    <th>Hôte du traitement</th>
  </tr>
  <?php foreach ($results as $filename => $data): ?>
    <tr>
      <td><?= e($filename) ?></td>
      <td><?= e(date('d/m/Y H:i:s', $data['created_at'])) ?></td>
      <td><?= e($data['result']['input']) ?></td>
      <td><?= e($data['result']['output']) ?></td>
      <td><?= e($data['result']['processed_by']) ?></td>
    </tr>
  <?php endforeach; ?>
</table>

<p><a href="form.php">Aller au formulaire</a></p>

<?php
require __DIR__ . '/inc/html_end.php';
```

form.php :

```
<?php

function doUppercase($input, $resultPath)
{
  // Simulate long processing...
  sleep(3);

  $output = mb_strtoupper($input, 'UTF-8');

  $result = array(
    'input' => $input,
    'output' => $output,
    'processed_by' => gethostname(),
  );

  $options = array('ftp' => array('overwrite' => true));
  $context = stream_context_create($options);

  file_put_contents($resultPath, json_encode($result), 0, $context);
}

const FOO_INPUTS_COUNT = 5;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  require_once __DIR__ . '/inc/config.php'; // for FOO_FILES_DIR
  $outDir = FOO_FILES_DIR;
  $uniq = uniqid('', true);
  for ($i = 0; $i < FOO_INPUTS_COUNT; $i++) {
    if (isset($_POST['inputs'][$i]) && $_POST['inputs'][$i] !== '') {
      doUppercase($_POST['inputs'][$i], "$outDir/$uniq-$i.json");
    }
  }

  // Redirect to the list page
  header('Location: list.php');
  exit;
}

$tpl_title = 'Formulaire';
require __DIR__ . '/inc/html_begin.php'; // uses $tpl_title, defines e()
?>

<h1>Formulaire de phrases</h1>

<form action="<?= e($_SERVER['SCRIPT_NAME']) ?>" method="post">
  <?php for ($i = 0; $i < FOO_INPUTS_COUNT; $i++): ?>
    <div><input type="text" name="inputs[]"/></div>
  <?php endfor; ?>
  <div><button type="submit">Envoyer</button></div>
</form>

<p><a href="list.php">Aller à la liste</a></p>

<?php
require __DIR__ . '/inc/html_end.php';
```

inc/config.php :

```
<?php

const FOO_FILES_DIR = 'ftp://foo:secret@files/uppercased';
```

inc/html_begin.php :

```
<?php

function e($var)
{
  return htmlspecialchars($var, ENT_COMPAT, 'UTF-8');
}

header('Content-Type: text/html; charset=UTF-8');
?>
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8"/>
  <title><?= e($tpl_title) ?> - Test Gearman</title>
  <link rel="stylesheet" href="style.css"/>
</head>
<body>
```

inc/html_end.php :

```
</body>
</html>
```

Les trois fichiers à inclure sont très simples. Précisons juste que le fichier inc/config.php définit le chemin d'un dossier accessible en lecture-écriture sur le serveur files. Notons que le fichier inc/html_begin.php lie une feuille de styles non reproduite ici. La page list.php affiche la liste des fichiers présents dans le dossier sur le serveur files, avec pour chacun le nom, la date de création (de modification) et le contenu (phrase originale, phrase transformée et nom de la machine ayant effectué le traitement).

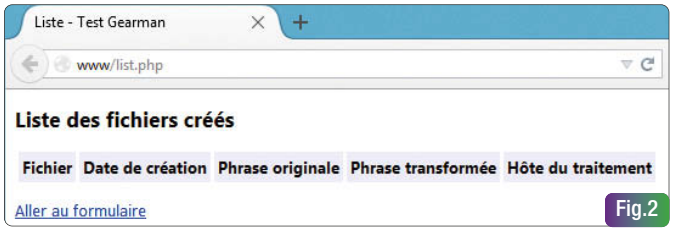
La page form.php a deux modes. Si on y accède directement (requête GET), la page affiche un formulaire de type « post » dont la cible est cette page elle-même, avec cinq champs de texte. Si on a soumis le formulaire (requête POST), pour chaque phrase soumise, le code exécute la fonction « doUppercase » définie plus haut, en lui passant aussi un chemin de fichier unique à créer sur le serveur files. Cette fonction effectue le traitement long et écrit un fichier de résultat. Enfin, le code redirige vers la page list.php.

Test de l'application

On ouvre un navigateur Web et on entre l'adresse « http://www/list.php » (ou « http://192.168.0.100/list.php ») pour afficher la page liste (Fig.2).

On voit que la liste est vide. On clique sur le lien pour accéder à la page formulaire. On saisit des phrases dans les cinq champs de texte et on soumet le formulaire. On remarque alors que la page reste bloquée en attente d'une réponse pendant que le traitement tourne, et c'est seulement au bout de 15 secondes (5 fois 3 sec) que la requête reçoit enfin une réponse qui redirige vers la page liste. On voit alors que cinq fichiers ont été créés au rythme d'un toutes les 3 secondes.

Suite le mois prochain



Page liste de notre application web

Le Drag and Drop en HTML 5

Le drag and drop (ou glisser-déposer) n'est pas nouveau dans le Web, mais pour le mettre en place, il a toujours fallu passer par des librairies Javascript plus ou moins complexes, car aucune API native n'existait. Depuis HTML 5, cette API est enfin arrivée et nous allons voir comment elle fonctionne.

Jérémy SCHERRER,
Développeur front-end chez **Netapsys**

Sachez qu'actuellement elle n'est pas très bien prise en charge (surtout sur les mobiles), et qu'il y a même encore des différences entre Chrome et Firefox, qui la comprennent pourtant tous les deux.
Je ne couvrirai pas les syntaxes particulières pour des navigateurs un peu trop anciens, mais vous montrerai le code minimum nécessaire pour faire fonctionner le Drag and Drop. A noter cependant qu'il existe d'autres attributs et propriétés additionnels pour agrémenter le tout.
Il y a 3 étapes de base à réaliser :

- Définir un élément déplaçable (c'est la source) ;
- Définir un élément capable de recevoir un autre élément (c'est la cible) ;
- Définir un comportement de déplacement et de dépôt via Javascript.

Plusieurs événements sont exploitables pour réaliser cela, parmi eux nous trouvons :

- *dragstart* : déclenché lors du début du déplacement d'un objet et actif pendant le déplacement ;
- *dragend* : déclenché lors de la fin du déplacement d'un objet ;
- *dragover* : déclenché en continu tant que la souris survole la cible ;
- *dragenter* : déclenché lors de l'entrée sur une cible avec un objet ;
- *dragleave* : déclenché lors de la sortie d'une cible ;
- *drop* : déclenché lors du dépôt d'un objet sur une cible.

Les étapes pour faire fonctionner le Drag and Drop

Etape 1 : Rendre un élément déplaçable

Il suffit d'ajouter l'attribut `draggable="true"` sur n'importe quel élément HTML. C'est tout !

```
<img src="photo1.jpg" alt="" draggable="true">
```

Etape 2 : Définir un élément récepteur des déplacements (une cible)

La cible peut être tout élément HTML auquel on attache les évènements relatifs à la réception d'un autre élément (`dragenter`, `dragleave`, `drop`).
Il suffit donc de lier ces événements à l'élément qui devra être la cible.

```
<div id="cible"></div>

<script>
var cible = document.getElementById('cible');

// ajout des gestionnaires d'événements
cible.addEventListener('dragenter', entreeCible);
cible.addEventListener('dragover', survolCible);
cible.addEventListener('dragleave', sortieCible);
cible.addEventListener('drop', deposer);

function entreeCible(event){
event.target.className = 'drop-me';
}

function sortieCible(event){
event.target.className = '';
  }
</script>
```

Notre élément HTML se comporte maintenant comme un récepteur de Drag and Drop (mais à ce stade rien ne se passe car nous n'avons pas défini l'évènement de dépôt).
Ce code ajoute une classe à la cible quand on la survole, et l'enlève quand on la quitte. Cela permet de changer l'apparence de la cible pour que l'on comprenne qu'il y a une interaction possible.

Le CSS :

Le CSS n'est pas obligatoire, mais il est très important de l'ajouter pour que l'utilisateur comprenne qu'il peut déplacer un objet, et que la cible peut le recevoir, car ce n'est pas évident sans aide visuelle.
Ainsi on peut montrer un curseur de déplacement, signaler la cible par un style spécifique, et surtout changer son style quand elle est survolée pour indiquer qu'on peut faire le dépôt.
Astuce : empêcher la sélection du texte de la source, sinon il sera difficile de sélectionner l'objet et non son texte. Cela se fait grâce à ce code :

```
[draggable=true] {
 -moz-user-select: none;
 -khtml-user-select: none;
 -webkit-user-select: none;
 user-select: none;
 cursor:move;
}
```

Etape 3 : Définir un comportement de déplacement et de dépôt

Il nous faut attacher un événement de déplacement à tous les éléments déplaçables.

```
// Création d'un objet NodeList contenant tous nos éléments déplaçables
var elementsDraggables = document.querySelectorAll('*[draggable=true]');

// Evènements 'dragstart' et 'dragend' pour chaque élément de la NodeList
[].forEach.call(elementsDraggables, function(element){
  element.addEventListener('dragstart', deplacement);
  element.addEventListener('dragend' , suppression);
});
```

La syntaxe particulière de la dernière fonction permet de convertir un objet

NodeList en Array pour le parcourir élément par élément. Ceci est indispensable car il n'est pas possible de parcourir un objet NodeList comme un Array.
L'API File et sa classe DataTransfer entrent ici en jeu pour améliorer l'expérience du Drag and Drop. Elle contient deux propriétés et une méthode :

- effectsAllowed
- dropEffect
- setDragImage()

effectsAllowed est utilisée pour changer le curseur de la souris en fonction du type d'action réalisé (copie de la source, déplacement de la source, lien vers la source...). C'est uniquement visuel.
dropEffect est équivalent mais du côté cible.
A l'heure actuelle, ces propriétés ne fonctionnent pas très bien (impossible de constater des différences au niveau du curseur de la souris, cela viendra certainement dans les futures versions des navigateurs).
setDragImage() permet de définir une image utilisée pour suivre la souris lors du déplacement (par défaut c'est l'image de la source en petit et semi-transparent).
La classe DataTransfer nous permet de définir et de stocker les données à retirer de la source, et de les transmettre à la cible. Cela peut être le texte de la source, la valeur de ses attributs, ses dimensions...).
Ceci est fait grâce à la méthode setData() qui prend 2 arguments : le type MIME des données transmises, et les données elles-mêmes.

```
function deplacement(event){
   // nous voulons faire une copie des données
   event.dataTransfer.effectAllowed='copy';
 // event.target représente l'objet source
 event.dataTransfer.setData('text/plain', event.target.innerText || event.target.textContent);
 }
```

Ici, nous copions le contenu texte de la source (un paragraphe par exemple) vers la cible. Tant que l'objet est en déplacement, il contient les données.
Firefox ne comprend pas innerText, et IE ne comprend pas textContent. Nous mettons donc les deux et laissons chacun interpréter ce qu'il comprend !

Récupérer les données transmises dans la cible :

La méthode getData() permet de récupérer les données envoyées par setData(). Nous l'utilisons donc sur la cible, lors du dépôt.

```
function deposer(event){
   event.preventDefault();
   // la cible doit être une copie de la source
   event.dataTransfer.effectAllowed=copy;
   // on récupère les données envoyées en text/plain
 event.dataTransfer.getData('text/plain');
 // dépôt fini : on redonne son apparence à la cible
 event.target.className = '';
 }
```

Nous appelons la méthode preventDefault(), ici car par défaut le comportement du navigateur n'incite pas à faire un dépôt, il faut donc l'annuler.
Le problème de ce code est que l'on peut uniquement récupérer du texte. Si les données envoyées sont du HTML, celui-ci ne pourra pas être exploité par la cible car elle n'accepte que du texte. On peut améliorer les choses en détectant le type de données reçues et en les gérant différemment.
De même lors de l'envoi des données, notre fonction deplacement() actuelle, envoie forcément du texte. Mais si elle est appliquée à une image par exemple, cela ne marchera pas car elle ne contient pas de texte. Nous pouvons donc ajuster nos fonctions de déplacement et de dépôt pour les rendre plus largement utilisables.

```
function deplacement(event){
   event.dataTransfer.effectAllowed='copy';
   var srcImage = event.target.getAttribute('src');

   // si la source a un attribut src, on envoie sa valeur (en recréant une nouvelle image)
   if(srcImage !== null){
      event.dataTransfer.setData('text/html', '<img src="'+srcImage+'">');
   }

   // si la source n'est pas une image, on envoie son texte
   else{
      event.dataTransfer.setData('text/plain', event.target.innerText || event.target.textContent);
   }
}
```

Bien entendu, une image n'est pas le seul élément HTML ne contenant pas de texte. Ce code peut donc être enrichi en testant les autres éléments non textuels.
Si nous voulons obtenir la valeur d'un attribut HTML quelconque, nous pouvons nous baser sur le code ci-dessous pour l'image, mais l'envoyer en text/plain.

Exemple d'amélioration :

```
function deposer(event){
   event.preventDefault();
   event.dataTransfer.effectAllowed=copy;

   // si on reçoit du texte, on l'affiche
   if (event.dataTransfer.getData('text/plain')){
      event.target.innerHTML += event.dataTransfer.getData('text/plain');
   }

   // si on reçoit du HTML, on l'affiche
   if (event.dataTransfer.getData('text/html')){
      event.target.innerHTML += event.dataTransfer.getData('text/html');
   }
   event.target.className = '';
}
```

Notre fonction accepte maintenant les 2 types de données. A noter qu'il existe aussi les formats text/uri-list, text et url.
Ce code permet de prendre une image ou un texte de la page définie comme déplaçable et de le copier dans la cible. Notre image sera donc à 2 endroits de la page.
Si nous voulons montrer que nous déplaçons notre élément de la source vers la cible, nous pouvons supprimer la source à la fin du déplacement. Nous aurons réellement l'impression d'un déplacement. Cela peut être utile si nous développons un jeu par exemple.
C'est la fonction associée à l'évènement dragend qui est concernée :

```
// Suppression de l'élément original
function suppression(event){
   event.target.parentNode.removeChild(event.target);
}
```

Notre élément source a maintenant disparu. Attention, si vous le faites, il est vivement conseillé de vérifier qu'il est bien dans la cible avant de le supprimer, car ce code le supprimera dans tous les cas, que le déplacement soit réussi ou non !

Enrichissement du Drag and Drop : évènement dragover

Il est déclenché périodiquement et en continu tant que la souris survole la cible. Une utilisation possible est d'associer un compteur à cet événement pour indiquer pendant combien de temps le survol est effectué. Dans le cas d'un jeu, le joueur peut être éliminé s'il est trop lent par exemple.

```
var dragoverCount = 1 ;

function survolCible(event){
   dragoverCount++ ;
   if(dragoverCount > 60){
      alert('Vous êtes trop lent, dépêchez-vous !) ;
   }
   event.preventDefault();
   return false;
}
```

Cas concret : Glisser un fichier de l'ordinateur vers le navigateur et l'exploiter

Une utilisation sympathique du Drag and Drop est de pouvoir faire glisser directement des fichiers depuis l'ordinateur vers le navigateur, sans passer par un champ de formulaire (un peu comme dans Gmail pour glisser des pièces jointes). Ici, seule la fonction de dépôt nous intéresse. Seule celle-ci sera modifiée car elle devra détecter si l'élément déposé est un élément HTML (donc déjà dans la page) ou un fichier (donc extérieur à la page).

```
function deposer(event){
   event.preventDefault();
   var fichiers = event.dataTransfer.files;
}
```

La propriété event.dataTransfer.files (de type FileList) contient la liste des fichiers déposés (de type File). Nous pouvons donc déposer plusieurs fichiers d'un seul mouvement de souris.
Nous pouvons alors accéder aux informations suivantes des fichiers : nom, taille, type, date de modification.

```
function deposer(event){
   event.preventDefault();
   var fichiers = event.dataTransfer.files;

   for(var i=0; i<fichiers.length; i++){
      event.target.innerHTML += 'Nom : ' + fichiers[i].name + '<br>';
      event.target.innerHTML += 'Taille (octets) : ' + fichiers[i].size + ' <br>';
      event.target.innerHTML += 'Type : ' + fichiers[i].type;
   }
   event.target.className = '';
}
```

Pour pouvoir vraiment lire les fichiers (accéder à leur contenu, pas à leurs informations), nous avons besoin de la classe FileReader de l'API File.
Il est conseillé de la tester avec une adresse en http:// et non en file://.
Nous devons créer un objet FileReader pour chaque fichier de notre liste.

Un objet FileReader contient plusieurs fonctions dont :
readAsText() pour lire les fichiers textes ;
readAsDataUrl() pour lire les fichiers binaires comme les images.

Lire un fichier texte déposé depuis l'ordinateur :

```
function deposer(event){
   event.preventDefault();
   var fichiers = event.dataTransfer.files;

   for(var i=0; i<fichiers.length; i++){
      chargerFichier(fichiers[i]);
   }

   event.target.className = '';
}

function chargerFichier(fichier){

   // création de l'objet FileReader
var reader = new FileReader();

   // quand le fichier est entièrement arrivé, il est contenu dans event.target.result
   reader.onload = function(event){
      // affichage du contenu du fichier dans la cible
dropzone.innerHTML += event.target.result + '<br>';
   };

   // lancement de la lecture du fichier
   reader.readAsText(fichier);
}
```

Lire un fichier binaire déposé depuis l'ordinateur :

La fonction deposer() est la même que pour un fichier texte. Seule la lecture du fichier va différer.

```
function chargerFichier(fichier){
   var reader = new FileReader();

   reader.onload = function(event){
      // création d'une balise image
      var img = document.createElement('img');
      // affectation du fichier comme source de la nouvelle image
      img.setAttribute('src', event.target.result);
      // ajout de l'image dans la zone cible
      dropzone.appendChild(img);
   };

   // lecture du fichier
   reader.readAsDataURL(fichier) ;
}
```

La lecture de l'objet en tant que DataUrl est possible grâce à la spécification Data URI, qui permet le stockage de données via une adresse virtuelle sous la forme data :. Le fichier binaire de base est en fait converti en texte du type :
data:[<mediatype>][;base64],<donnees>
C'est le même principe que les images encodées en base64 : le contenu devient une longue chaîne de texte et cela permet de « reproduire » le fichier.

Là encore, notre script peut être amélioré car la fonction chargerFichier() gère soit exclusivement un fichier texte (exemple 1), soit exclusivement un fichier binaire (exemple 2). Idéalement une unique fonction chargerFichier() doit détecter le type de fichier glissé et le gérer comme il faut.

Exemple d'amélioration :

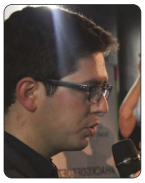
```
function deposer(event){
  event.preventDefault();
  var fichiers = event.dataTransfer.files;


  for(var i=0; i<fichiers.length; i++){
      // si le fichier est une image
      if((/\.(gif|jpg|jpeg|tiff|png)$/i).test(fichiers[i].name)){
          // on appelle la fonction avec un second paramètre
          chargerFichier(fichiers[i], true);
      }

// sinon on l'appelle sans second paramètre
      else{
          chargerFichier(fichiers[i]);
      }
  }

  event.target.className = '';
}



function chargerFichier(fichier, isImage){
  var reader = new FileReader();

  // si le second argument est défini, c'est une image, on gère le fichier en tant que tel
  if(isImage){
      reader.onload = function(event){
          var img = document.createElement('img');
          dropzone.appendChild(img);
          img.setAttribute('src', event.target.result);
          dropzone.innerHTML += '<br>';
      };

      reader.readAsDataURL(fichier);
  }

  // sinon c'est un fichier texte, on le gère comme un texte
  else{
      reader.onload = function(event){
          dropzone.innerHTML += event.target.result + '<br>';
      };

      reader.readAsText(fichier);
  }
}
```

Notre fonction chargerFichier() peut maintenant gérer ces 2 types de fichiers. En conclusion, j'espère que cette présentation vous aura donné envie de jouer avec le Drag and Drop. Celui-ci peut servir autant dans une application Web comme substitution d'un champ de formulaire, que dans les jeux par exemple. Cependant, gardez bien à l'esprit la faiblesse actuelle du support navigateur si vous l'utilisez. Je remercie Rodolphe Rimelé pour son excellent livre *HTML5 : une référence pour le développeur Web* qui m'a été utile pour cet article. Si vous souhaitez en savoir plus, vous pouvez également consulter le site de Jakob Jenkov et ces 2 pages :

http://tutorials.jenkov.com/html5/drag-and-drop.html
http://tutorials.jenkov.com/html5/file-api.html

Interconnecter vos objets avec Node.js 2eme partie

En permettant le développement d'applications côté serveur en Javascript, langage connu pour sa grande souplesse, Node.js offre la possibilité de manipuler les couches hautes comme les couches basses. Ce sont ces dernières qui permettent de prendre le contrôle des objets et de créer entre eux des interactions. Après avoir défini un environnement adéquat, on peut alors obtenir toutes les interactions induites par la potentialité des objets.



Matthieu Bouilloux
Développeur web et applicatif. Intéressé par tout type de nouvelle technologie tel que l'Iot ou encore l'analyse des marchés boursier.
http://www.katlea-edition.com
matthieu@katlea-edition.com

LANCER NODE WEBKIT SUR LA SORTIE VIDÉO DU PI 2

D'abord installer un environnement minimal (fluxbox) :

```
sudo apt-get install fluxbox
```

Puis on change de gestionnaire de fenêtre :

```
sudo update-alternatives --config x-window-manager
sudo update-alternatives --install "/usr/bin/x-session-manager" "x-session-manager"
 "/usr/bin/startfluxbox" 99
sudo update-alternatives --config x-session-manager
```

Choisir l'option startfluxbox, puis on insère une alternative pour le session-manager, on rechoisit l'option startfluxbox et on redémarre le Pi 2 sudo reboot

Ajouter un fond d'écran et un splash screen

On copie via sftp une image en "jpg" vers le dossier ~/background créé au préalable. Ensuite :

```
nano ~/.fluxbox/startup

> Dans le fichier startup après xmodmap "/home/pi/.Xmodmap"
> Ajouter :

fbsetbg - C ~/background/mon_image.jpg

nano ~/.fluxbox/overlay

> Remplacer : "! background: none" par :
background.pixmap: ~/background/mon_image.jpg
```

Restreindre l'interface

Remplir le fichier init de fluxbox comme suit :

```
nano ~/.fluxbox/init

> session.screen0.toolbar.visible: false
> session.screen0.iconbar.mode: {static groups} (workspace)
> session.screen0.tabs.usePixmap: false
> session.screen0.slit.placement: RightBottom
> session.screen0.slit.direction: Vertical
> session.configVersion: 13
```

Vider les fichiers menu et keys en en conservant une copie :

```
cp ~/.fluxbox/menu ~/.fluxbox/menu.back
cp ~/.fluxbox/key ~/.fluxbox/key.back
echo > ~/.fluxbox/menu
echo > ~/.fluxbox/key
```

On rajoute un serveur VNC pour visualiser le rendu depuis l'écran de l'ordinateur :

```
sudo apt-get install x11vnc
x11vnc -storepasswd "VOTRE MOT DE PASSE" ~/.vnc_passwd
x11vnc -many -rfbauth ~/.vnc_passwd
```

Vous n'avez plus qu'à vous connecter à raspberrypi.local avec un client VNC pour voir le résultat !

Installer node webkit

Activer le mode Desktop :

```
sudo raspi-config

Sélectionner les options suivantes :
> Boot options
> B4 Desktop Autologin
```

Télécharger et installer node webkit dans le PATH :

```
cd ~
wget https://github.com/jtg-gg/node-webkit/releases/download/nw-v0.12.0/nwjs-v0.
12.0-linux-arm.tar.gz
tar -zxvf nwjs-v0.12.0-linux-arm.tar.gz
rm -zxvf nwjs-v0.12.0-linux-arm.tar.gz
mv nwjs-v0.12.0-linux-arm nwjs
echo 'PATH=$PATH:~/nwjs' >> ~/.bashrc
```

Lancer node webkit

Il faut pour cela que le Pi 2 soit connecté à un écran. Il faut ensuite indiquer l'écran dans la ligne de commande juste avant de lancer nw

```
DISPLAY=:0 nw /repertoire_du_projet_webkit
```

Lancer webkit au démarrage

Il faut ajouter la ligne précédente dans le fichier rc.local
Attention à bien rajouter la ligne avant le exit 0

```
sudo nano /etc/rc.local

> DISPLAY=:0 nw /repertoire_du_projet_webkit
> exit 0
```

Et après

Vous pouvez contrôler tout l'affichage de l'écran grâce à node-webkit (tel un Node.js OS), le modifier avec des interactions de type serveur, Bluetooth ou autre !

MAPPAGE DU BLUETOOTH D'UN MYO AVEC UNE EDISON

Le bracelet Myo est un électromyogramme avancé qui se connecte en Bluetooth. Je vais vous expliquer le principe de ses fonctionnalités avec le contrôleur Bluetooth intégré de la carte Intel Edison. Cette étape permettra de recréer une interface logicielle fonctionnant sous Linux. Nous aurons besoin de la spécification du protocole Bluetooth du Myo pour réaliser le mappage. Pour ce faire, nous avons à notre disposition un fichier header en C++ fourni par ThalmicLabs, les créateurs du Myo : https://github.com/thalmiclabs/myo-bluetooth
et aussi du module noble pour Node.js sur l'Edison :

```
npm install noble
```

Tous les Myo émettent le même identifiant. On peut donc se connecter aisément au Myo le plus proche grâce à cette chaîne de charactères : 'd5060001a904deb947482c7f4a124842'.
Une fois passée la fonction basique de connexion, on attaque le protocole Bluetooth Low Energy (4.0). Il est composé de services possédant des classificateurs d'évènements qui sont eux-mêmes composés de caractéristiques. Chaque caractéristique peut avoir une fonctionnalité "read", "write" et/ou "notify".
La fonctionnalité "read" permet de lire la valeur d'une caractéristique sous forme de Buffer. A l'inverse, la fonctionnalité "write" écrit une valeur (aussi sous forme de buffer) pour une caractéristique. Quant à la fonction "notify", elle permet d'obtenir de manière asynchrone le changement de statut d'une ou de plusieurs valeurs de façon continue sur la fonctionnalité "read".
A partir de ces diverses fonctionnalités du Myo, j'ai analysé le fonctionnement de communication Bluetooth de bas niveau, puis j'ai associé le module noble en créant des fonctions simples d'interactions.
Pour ce faire, j'ai utilisé l'Edison car son architecture inclut le Bluetooth, et il est facile d'y installer des paquets Node.js.
Ensuite j'ai créé un script par-dessus le module noble pour communiquer de manière simplifiée avec le Myo, accédant ainsi aux mouvements de mon bras et de ma main.
La documentation et le script sont disponibles sur Github :
https://github.com/katlea/node-myo-edison
Pour plus d'éléments sur l'installation :
http://www.instructables.com/id/MyoCraft-Myo-Armband-with-nodejs-on-Intel-Edison-B

MODULES D'ENCEINTES WIFI AVEC TÉLÉCOMMANDE

Prérequis

Un ou plusieurs Raspberry Pi avec dongle WiFi et Node.js installés
Un ordinateur ou board servant de serveur.
Une board Edison / Yùn pour la télécommande

Partie Rapsberry Pi

Il faut installer un player audio acceptant le streaming :

```
sudo apt-get install mplayer2
```

Nous allons faire du streaming depuis Node.js vers le mplayer. Pour l'envoi des données de la musique, nous allons créer un pipe (c'est-à-dire envoyer des informations d'un processus à un autre) au travers d'un flux standard nommé STDIN (canaux pour l'entrée de données sur les systèmes d'exploitation UNIX). Pour contrôler le mplayer, le pipe sur STDIN étant déjà occupé par le flux de la musique, nous allons créer un fichier FIFO. Il s'agit d'un fichier semblable à un pipe qui, au lieu d'être une connexion temporaire, a une présence physique et un nom.
L'action du processus mplayer sera d'ouvrir le fichier FIFO créé pour communiquer avec lui.

Préparation

Préparer le fichier app.js avec les requires :

```
var fs = require('fs');
var cp = require('child_process');
var spawn = cp.spawn;
var path = require('path');
```

Puis créer, si celui-ci n'existe pas, le fichier FIFO :

```
if(!fs.existsSync('./fifo')){
  fs.mkdirSync('./fifo');
}
if(!fs.existsSync('./fifo/control')){
  var mkfifoProcess = spawn('mkfifo',  ['./fifo/control']);
  mkfifoProcess.on('exit', function (code) {
    if (code == 0) {
      console.log('fifo created: ' + './fifo/control');
    } else {
      console.log('fail to create fifo with code:  ' + code);
    }
  });
}
```

Aperçu de streaming local

Commencer par ajouter un fichier music.mp3 au même endroit que le fichier app.js
Dans le fichier app.js pour tester le Raspberry, on instancie un stream de lecture :

```
var str = fs.createReadStream(file);
```

On utilise la fonction spawn de node.js qui permet d'exécuter un processus enfant en continu et de manière asynchrone :

```
var mplayer = spawn('mplayer', ['-quiet', '-input', 'file=' + path.resolve('.') + '/fifo/control',
 '-'], { stdio: 'pipe' });
```

L'option quiet évite les écritures parasitaires dans la console. L'option input permet d'indiquer le fichier FIFO au mplayer. Pour cette dernière option, il faut préciser le chemin absolu grâce à la fonction path. L'option -, qui doit intervenir en dernier, active le streaming. Enfin on indique à node.js que les entrées et les sorties se font selon un pipe. Si l'on oublie cette précision, la lecture du streaming s'arrêtera après un certain volume d'informations. On envoie ensuite le stream du fichier mp3 au mplayer.

```
str.pipe(mplayer.stdin);

> OU <

str.on('data', function (data) {
  mplayer.stdin.write(data);
}
```

Exécuter une commande du mplayer durant le streaming

Il faut écrire dans le fichier FIFO. Pour cela on exécute un simple echo. Attention, le chemin du fichier FIFO doit être absolu (d'où le path.resolve('.')).

Exemples :

Mettre en pause ou relancer la lecture :

```
cp.exec('echo "pause" > '+path.resolve('.')+'/fifo/control\n');
```

Modifier le volume (où "100" est le volume maximal)

```
cp.exec('echo "volume 100 1" > '+path.resolve('.')+'/fifo/control\n');
```

Installer un client socket.io

Dans le répertoire du fichier app.js, installer le client :

```
sudo npm install socket.io-client
```

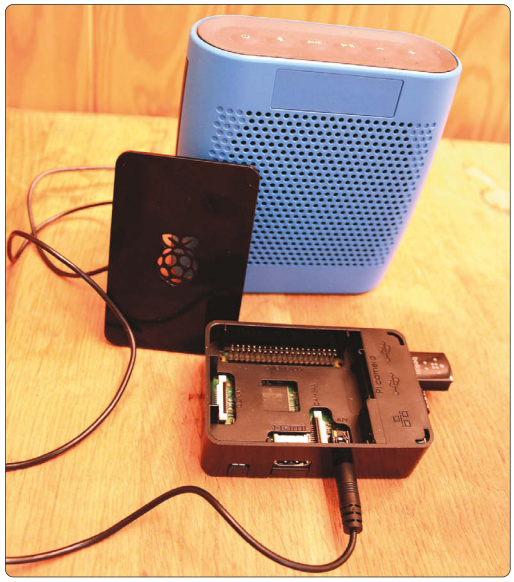
Les sockets sont un moyen simple d'établir une communication entre les processus node.js des différents devices. Il vous suffit de vous connecter au serveur (voir la suite de l'article pour le script du serveur).

```
var socket = require('socket.io-client')('http://NOM_DU_SERVEUR.local:3001');
```

Sous Windows pour trouver le hostname de votre ordinateur, ouvrez l'invite de commande (touche Windows + touche R, exécuter : cmd), et tapez hostname. Sous Linux dans la console : hostname
Remarque : sous Windows vous devez avoir le logiciel "bonjour" de Apple installé (lien ci-dessous).
https://support.apple.com/kb/dl999

Script complet du Raspberry Pi :

```
var fs = require('fs');
var cp = require('child_process');
var spawn = cp.spawn;
var path = require('path');

if(!fs.existsSync('./fifo')){
  fs.mkdirSync('./fifo');
}

if(!fs.existsSync('./fifo/control')){
  var mkfifoProcess = spawn('mkfifo',  ['./fifo/control']);
  mkfifoProcess.on('exit', function (code) {
    if (code == 0) {
      console.log('fifo created: ' + './fifo/control');
    } else {
      console.log('fail to create fifo with code:  ' + code);
    }
  });
}

var socket = require('socket.io-client')('http://NOM_DU_SERVEUR.local:3001');
socket.on('stream', function(data){
  mplayer.stdin.write(data);
});
socket.on('pause', function(){
  cp.exec('echo "pause" > '+path.resolve('.')+'/fifo/control\n');
});
socket.on('volume', function(data){
  cp.exec('echo "volume '+data+' 1" > '+path.resolve('.')+'/fifo/control\n');
});
```



Le Raspberry est prêt à recevoir les données via le serveur socket.io relié arbitrairement au port 3001 et à exécuter des commandes basiques, telles que : le volume / play-pause. A exécuter avec les droits administrateurs (sudo);

Le script serveur

```
var fs = require('fs');

// Création du serveur HTTP auquel on vient greffer socket.io
// Celui-ci sera le serveur utilisé pour le Raspberry pi
var app = require('http').createServer(function(req, res){
  res.send('ok');
});
var io = require("socket.io")(app);
app.listen(3001);

// Création du deuxième serveur HTTP pour communiquer avec l'Arduino Yùn
var app2 = require('http').createServer(function(req, res){
  res.send('ok');
});
var io2 = require("socket.io")(app2);
app2.listen(3002);

var pool = {}; // Objet qui va accueillir les connections sockets du/des Raspberry
var play = false; //Etat de la lecture

// A la connexion d'un Raspberry on ajoute le socket à l'objet pool
// Et Inversement à la déconnexion
io.on('connection', function(socket){
  pool[socket.id] = socket;
});
io.on('disconnect', function(socket){
  delete pool[socket.id];
});

// Fonction pour déclencher le streaming vers les Raspberry
function streamIt() {
  // Lecture du fichier son
  var str = fs.createReadStream('./music/'+files[file]);

  // Envoi des données au Raspberry
  str.on('data', function (data) {
```

```
    for (var k in pool) {
      pool[k].emit('stream', data);
    }
  });
}

// Partie socket Arduino pour la télécommande
// On récupère la commande et on la dispatche sur les Raspberry Pi
io2.on('connection', function(socket){
  console.log('arduino');
  socket.on('volume', function(data){
    for (var k in pool) {
      pool[k].emit('volume', data);
    }
  });
  socket.on('pause', function(){
    console.log('play');
    if(play) {
      for (var k in pool) {
        pool[k].emit("pause");
      }
    } else {
      play = true;
      streamIt();
    }
  });
});
```

L'Arduino Yùn comme télécommande

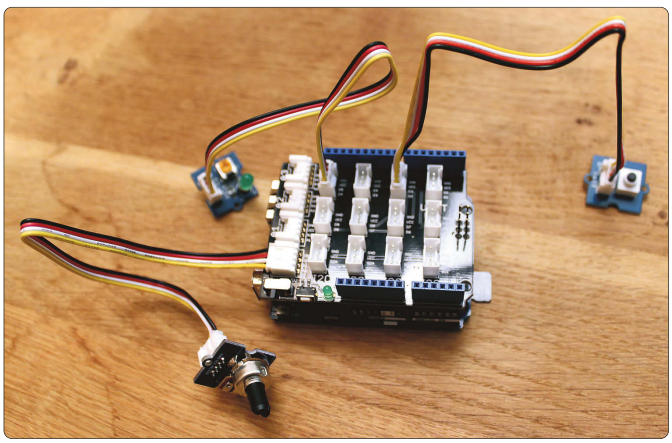
Une fois le Yùn opérationnel (avec cylon.js), il faut installer un client socket.io. Sur votre ordinateur :

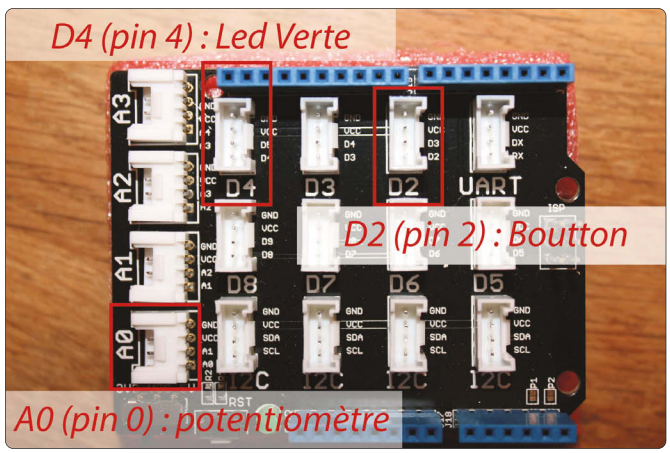
```
npm install socket.io-client
```

Puis coller le dossier socket.io-client dans le dossier node_modules de l'application de l'Arduino. Le script de la télécommande :

```
var Cylon = require('cylon');

var control = {};

Cylon.robot({
  // Connexion à l'Arduino
  connections: {
    arduino: { adaptor: 'firmata', port: '/dev/ttyATH0' }
  },

  // On ajoute les sensors : led de status de connexion,
  // bouton pause-play,
  // Potentiomètre pour le son.
  devices: {
    led: { driver: 'led', pin: 4 },
    play: { driver: 'button', pin: 2 },
    sensor: { driver: 'analog-sensor', pin: 0, lowerLimit: 0, upperLimit: 700 }
  },

  work: function(my) {
    // Pour l'adresse du serveur, utiliser cette fois-ci son IP.
    var socket = require('socket.io-client')('http://XXX.XXX.XXX.XXX3002');

    // A la connexion on allume la led
    socket.on('connect', function(){
      my.led.toggle();
    });

    // Et inversement à la déconnexion
    socket.on('disconnect', function(){
      my.led.toggle();
    });

    // On envoie le Play / Pause
    my.play.on('push', function(){
      socket.emit('pause');
    });

    // On envoie le niveau de volume en fonction du potentiomètre
    var volume = null;
    setInterval(function(){
      var newValue = parseInt(my.sensor.analogRead() / 7);
      if(newValue != volume){
        socket.emit('volume', newValue);
        volume = newValue;
      }
    }, 500);
  }
}).start();
```

Bon streaming !





.Net Native : proche du C++ ?

Dans les années 90, Microsoft déploie son propre environnement d'exécution managé appelé .Net. et propose aux développeurs un langage de haut niveau, leur permettant de se concentrer sur les fonctionnalités de leurs applications plus que sur les subtilités de gestion de la mémoire. Néanmoins, .Net n'a jamais été aussi léger et rapide qu'un programme écrit dans un langage bas niveau.

Jérôme GIACOMINI
Jonathan ANTOINE
Consultants Infinite **Square**
Blog : http://blogs.infinitesquare.com/

INFINITE SQUARE

Avec l'arrivée des tablettes et téléphones, les performances redeviennent primordiales et Microsoft se devait de proposer un environnement d'exécution adapté à la mobilité.

Avec .Net Native, Microsoft réunit le meilleur des deux mondes, facilité d'écriture du code managé et rapidité du code natif. Les développeurs familiarisés à .Net pourront utiliser l'intégralité de leurs compétences en C# ou Visual Basic. Une fois compilées avec .Net Native, les applications démarreront et s'exécuteront plus vite et ne nécessiteront plus l'installation de .Net Framework sur le poste client pour s'exécuter.

Pourquoi .Net Native ?

Microsoft a créé .Net Native pour :

- Avoir des meilleures performances qu'en .Net;
- Réduire l'empreinte mémoire;
- Être indépendant du système d'exploitation et des mises à jour de celui-ci;
- Ne pas perdre en productivité en utilisant un langage haut niveau.

Et c'est disponible ?

Les applications du Windows Store pour Windows 10 n'utilisent que .Net Native. Microsoft prévoit de créer une version du compilateur pour les applications desktop et serveur.

Comment fonctionne .Net Native?

Les processus de compilation et d'exécution sont différents entre .Net et .Net Native. Ces différences expliquent facilement les gains de performances apportés par .Net Native.

.Net :

A la compilation d'une application .Net :

- Le code C# est transformé en code IL (*Intermediate Language*).

Ensuite, l'exécution du programme .Net comprend trois étapes :

- Le code IL est chargé en mémoire;
- Compilation par le compilateur JIT (*Just in time*) en code natif;
- Exécution du code natif.

.Net Native :

La compilation .Net Native :

- Transforme le code C# en code IL (*Intermediate Language*);
- Puis il est compilé en code Natif :
  - Supprime toutes les classes non utilisées;
  - Applique diverses optimisations;
  - Compile en code natif.

L'exécution du code .Net Native est quant à elle plus simple que .Net :

- Charge le code natif en mémoire ;
- Exécution du code natif.

Il en résulte que le démarrage d'une application .Net Native est 60% plus rapide qu'une application .Net. Fig.1.

> **Note :** *La compilation .Net Native est plus lente qu'en .Net. Pour pallier cet inconvénient, le mode « Debug » permet d'exécuter sans compiler en code natif, ce qui accélère le process de développement.*

Pourquoi .Net Native est-il plus performant que .Net ?

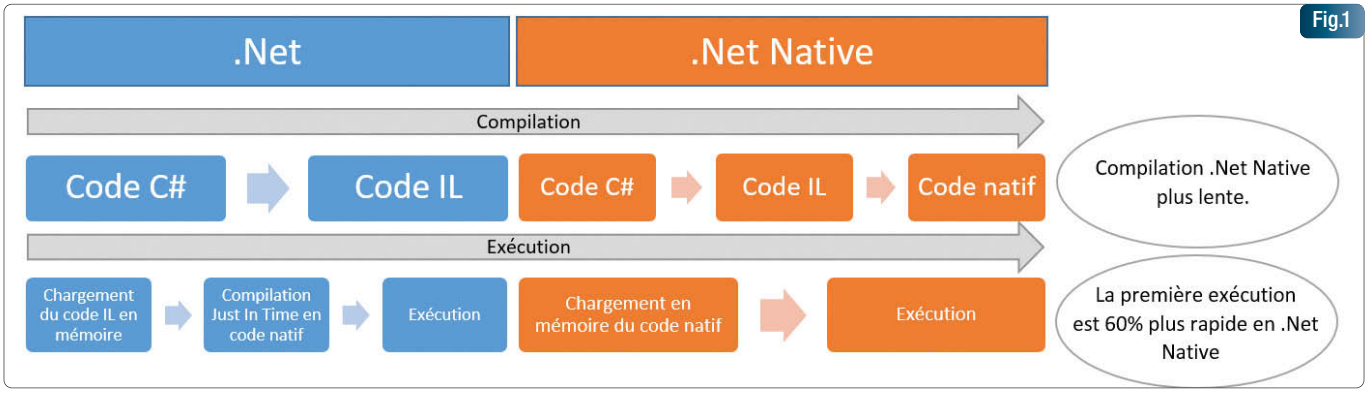
Optimisations dues au compilateur C++

Le compilateur de .Net Native utilise les dernières versions des compilateurs C++; il bénéficie donc de tous les avantages de ces compilateurs. De ce fait, les calculs arithmétiques sont beaucoup plus rapides en .Net Native. Nous avons choisi de tester les performances .Net Native avec une fonction qui convertit les couleurs d'une image en niveaux de gris. Nos tests ont confirmé que la fonction .Net Native est 2 fois plus rapide que la même fonction .Net.

Ci-dessous le code de cette fonction :

```
for (int i = 0; i < srcPixels.Length; i += 4)
{
double b = srcPixels[i] / 255.0;
      double g = srcPixels[i + 1] / 255.0;
      double r = srcPixels[i + 2] / 255.0;
      byte a = srcPixels[i + 3];
```

Fig.1



```
    double e = (0.21 * r + 0.71 * g + 0.07 * b) * 255;
    byte f = Convert.ToByte(e);
  dstPixels[i] = f;
  dstPixels[i + 1] = f;
  dstPixels[i + 2] = f;
  dstPixels[i + 3] = a;
}
```

Dans cet exemple, le gain de performance est dû au fait que le compilateur .Net Native a utilisé l'auto-vectorisation des calculs dans la boucle For. C'est-à-dire que le compilateur C++ a adapté automatiquement les traitements à des processeurs vectoriels (SIMD). Le compilateur JIT (Just in time) de .Net est très en retard sur cette partie-là.

Optimisations dues à la dé-virtualisation

Lors de la compilation, .Net Native applique deux techniques d'optimisation appelée dé-virtualisation :

- InterfaceDevirtualization : élimine l'appel à l'interface lors de la compilation et remplace par un appel direct à la méthode.
- AbstractClassDevirtualization : le même genre d'optimisation est appliqué sur les classes héritant de classes abstraites.

Ces deux optimisations lors de la compilation permettent d'effectuer une instruction en moins à chaque exécution de méthode d'une interface/classe abstraite.

Exemple de dé-virtualisation :

```
IMonInterface monInstance = new MaClasse();
monInstance.MaMethode();
```

En .Net l'exécution comprend 2 étapes :

- Résolution de la méthode « MaMéthode » permettant de retrouver l'implémentation réelle de la méthode.
- Exécution de la méthode.

En .Net Native, la première étape de résolution de méthode n'existe pas. La compilation .Net Native supprime l'appel à l'interface et envoie directement à la méthode implémentée.

Une empreinte mémoire plus petite

Les exécutables .Net Native utilisent 15 à 20% de mémoire en moins.

Lors de l'étape de compilation, le compilateur de .Net Native analyse votre application et détecte tout le code qui peut être supprimé (étape 2b). L'exécutable est donc plus petit et a une empreinte mémoire plus petite car il charge en mémoire moins de code.

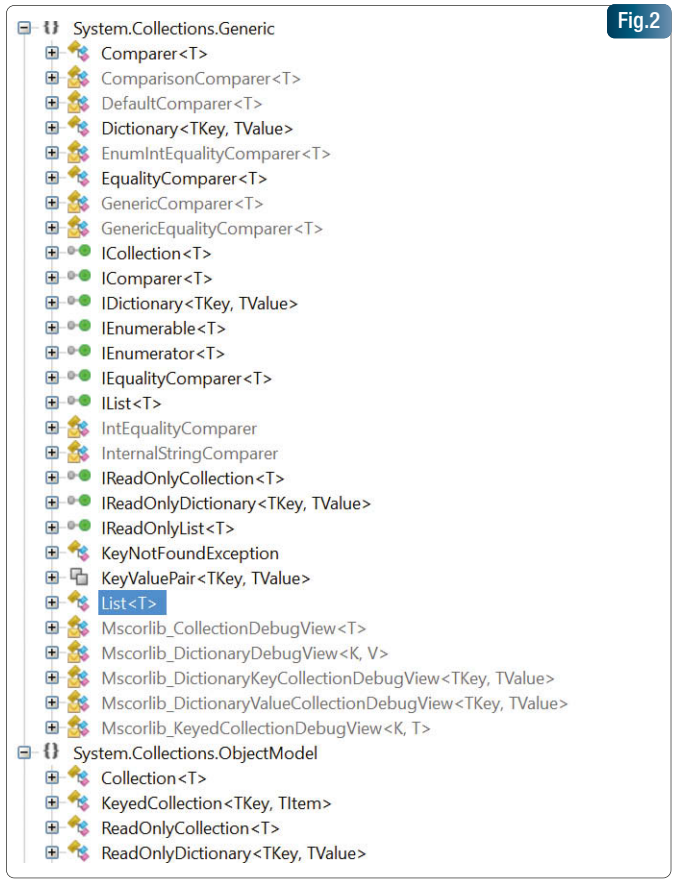
Par exemple, en .Net, si on utilise uniquement la collection List<T> , le programme chargera en mémoire le code de plus d'une centaines de classes qui ne seront pas utilisées dans la plupart des cas. Lors de la compilation, .Net Native supprime toutes les classes non utilisées par le programme. Fig.2.

Des exécutables indépendants

Contrairement aux exécutables .Net, les exécutables .Net Native embarquent tout le code dont il est dépendant. Ils sont donc indépendants des mises à jours du système d'exploitation et peuvent utiliser au plus vite les nouvelles fonctionnalités/corrections/optimisations introduites dans les mise à jours des SDK. C'est dans la continuité de la stratégie de Microsoft de délivrer plus souvent des mises à jour de leurs produits. Il est à noter que le résultat d'une compilation produit 3 assemblées différentes :

- *<nom_application>*.exe - point d'entrée de l'application ;

Fig.2



- *<nom_application>*.dll - ensemble des classes de l'application ;
- mrt100_app.dll - le runtime factorisé que l'application utilise.

Contrairement aux assemblies .Net, celles-ci ne sont pas décompilables, donc plus besoin d'obfuscateur de code.

Et ça fonctionne de la même façon que .Net ?

.Net Native utilise les mêmes classes que .Net et met à disposition le même Garbage Collector.

Pour être plus léger, le .Net Native n'embarque pas les métadonnées de type .Net. Les opérations de réflexion et sérialisation utilisent ces métadonnées de type. Afin de rendre disponibles ces opérations, il est nécessaire de préciser au compilateur .Net Native qu'il doit « embarquer » les métadonnées. Un fichier .xml permet de guider le compilateur pour qu'il puisse embarquer les métadonnées précisées par le développeur.

Exemple de fichier rd.xml :

```
<Directives xmlns="http://schemas.microsoft.com/netfx/2013/01/metadata">
 <Application>
  <Namespace Name="MonAppNative.Services" Serialize="Required Public" />
  <Namespace Name="MonAppNative.ViewModels" Serialize="Required Public" />
  <Namespace Name="MonAppNative.DataModel" Serialize="Required Public" />
  <Namespace Name="System.Collections.ObjectModel" >
   <TypeInstantiation Name="ObservableCollection"
      Arguments="MonAppNative.ImageModel" Serialize="Public" />
  </Namespace>
 </Application>
</Directives>
```

.Net Native s'intègre dans la vision globale de Microsoft qui prévoit d'augmenter la fréquence de livraison de ses outils et d'être adapté aux usages futurs de l'informatique qui sont l'IoT et la mobilité.

Construire son drone

2e partie

Dans la première partie, je vous ai présenté mes choix en matière de mécanique du drone, à savoir le châssis, les moteurs, les hélices et les ESC (pour Electronic Speed Control). Je vous propose aujourd'hui de nous pencher sur ce qui permet au drone de voler et d'être autonome, avec son intelligence embarquée, ce qu'on appelle un "Contrôleur de Vol"; et de profiter d'un focus sur un autre élément essentiel : le duo Récepteur / Radiocommande.

François Chevalier
Concepteur développeur chez SQLI Enterprise à Nantes depuis 2011. Passionné par les nouvelles technologies et l'aéronautique, il s'intéresse au concept du Do It Yourself et essaye de l'appliquer dans son quotidien.

Le contrôleur de vol, l'intelligence du drone

Pour bien comprendre le rôle du contrôleur de vol dans notre processus, revenons brièvement sur le fonctionnement d'un drone et notamment les règles physiques qui le font voler : dans le cas d'un drone de type multicopter, c'est la rotation de chacune des hélices qui, grâce à la portance qu'elle génère, exerce une poussée vers le haut. Puisque le pas de l'hélice est fixe, la portance varie en fonction de la vitesse de rotation de celle-ci. Prenons un exemple simple pour illustrer ce principe : lorsque l'on veut que notre drone avance, les hélices qui se trouvent à l'avant du drone doivent avoir une vitesse de rotation légèrement inférieure aux hélices se trouvant à l'arrière. De ce fait, la partie avant du drone s'incline vers le bas ; la portance, en plus d'infliger une poussée verticale, applique également une légère poussée vers l'avant et amorce le déplacement voulu.

Gardons donc à l'esprit que les mouvements de notre drone seront liés à la vitesse de rotation de chacune de nos hélices et donc de chacun de nos moteurs.

Le pilote pourrait essayer, dans le cas d'un quadcopter qui ne comporte que quatre moteurs, de commander chacun d'eux manuellement, mais on imagine aisément la gymnastique intellectuelle de l'exercice et je vous laisse imaginer la rapidité d'exécution nécessaire pour 6, 8 voire 10 moteurs. C'est pour pallier cette complexité que le contrôleur de vol intervient. Ce module permet en effet de commander l'ensemble des moteurs à notre place, pour traduire auprès des moteurs les mouvements que l'on souhaite donner à notre drone. Autre avantage, il permet de maintenir seul l'assiette (c'est-à-dire de garder le drone à peu près horizontal de façon à ce qu'il se stabilise) du drone sans que le pilote intervienne sur les commandes. Ce comportement est possible car le contrôleur est équipé d'un accéléromètre qui lui indique son assiette et qui corrige les mouvements du drone en conséquence.

Ces caractéristiques basiques (multi-commande des moteurs et maintien automatique de l'assiette) font que l'on peut considérer le contrôleur de vol comme l'intelligence du drone.

Il s'agit certes d'une intelligence limitée mais elle joue un rôle déterminant en matière d'expérience de pilotage.

Outre ces caractéristiques de base, certains contrôleurs offrent des possibilités plus avancées, comme programmer une navigation via un circuit de points GPS ainsi qu'une altitude à suivre pour chaque point transmis. Dans ce cas, le drone est capable de façon autonome de décoller, puis d'aller d'un point à un autre en suivant le chemin qu'on lui a défini, puis d'atterrir. Pour permettre cette autonomie, il faut connecter au contrôleur un module GPS qui connaît sa position en temps réel. Attention toutefois, ce mode n'est pas assez autonome pour détecter les obstacles...

Certains contrôleurs sont également équipés d'une boussole leur permettant de situer l'orientation du drone par rapport au Nord magnétique. Cette information peut s'avérer utile si l'on souhaite que le drone tourne de façon automatique autour d'un point fixe en maintenant l'avant toujours en direction du point fixe.

Enfin, notons parmi les options possibles que d'autres contrôleurs sont capables de maintenir une altitude, de revenir seuls à leur point de départ, de décoller sans avoir à gérer la manette des gaz, ou encore de se mettre dans un mode "FailSafe", qui se déclenche lors d'une perte de communication avec la radiocommande ou la station au sol (dans ce cas le drone s'arrête ou revient à son point de départ).

Quelques contrôleurs du marché :

Maintenant que vous connaissez parfaitement les principes de base du contrôleur de vol, il est temps de parler de ce qui est disponible sur le marché et de quel contrôleur choisir en fonction de son besoin. Ci-dessous quatre contrôleurs que je considère comme les principaux :

- **CC3D / Naze32 :** un contrôleur très petit, idéal pour un drone de type FPV Racing qui demande de la maniabilité et de la nervosité. Il fait partie des contrôleurs les moins chers du marché. Il est open source et a été créé par la communauté Openpilot. On le trouve sous différentes formes où les connecteurs peuvent être à souder soi-même pour les cartes les moins chères (25 - 65 €). Il n'y a pas de boussole intégrée mais il est possible d'y connecter un GPS avec boussole. Des logiciels open source sont disponibles sur Internet pour pouvoir configurer le contrôleur et c'est assez simple à utiliser.
- **DJI Naza :** un très bon contrôleur de la marque DJI, qui fabrique les drones du même nom. C'est un contrôleur fiable et précis, qui permet de réaliser l'ensemble des tâches qu'un contrôleur peut offrir lorsqu'il est associé à un GPS. Il a toutefois l'inconvénient d'être cher (150 - 250 €) et les possibilités de personnalisation ou de développement sont quasi inexistantes car il n'est pas open source

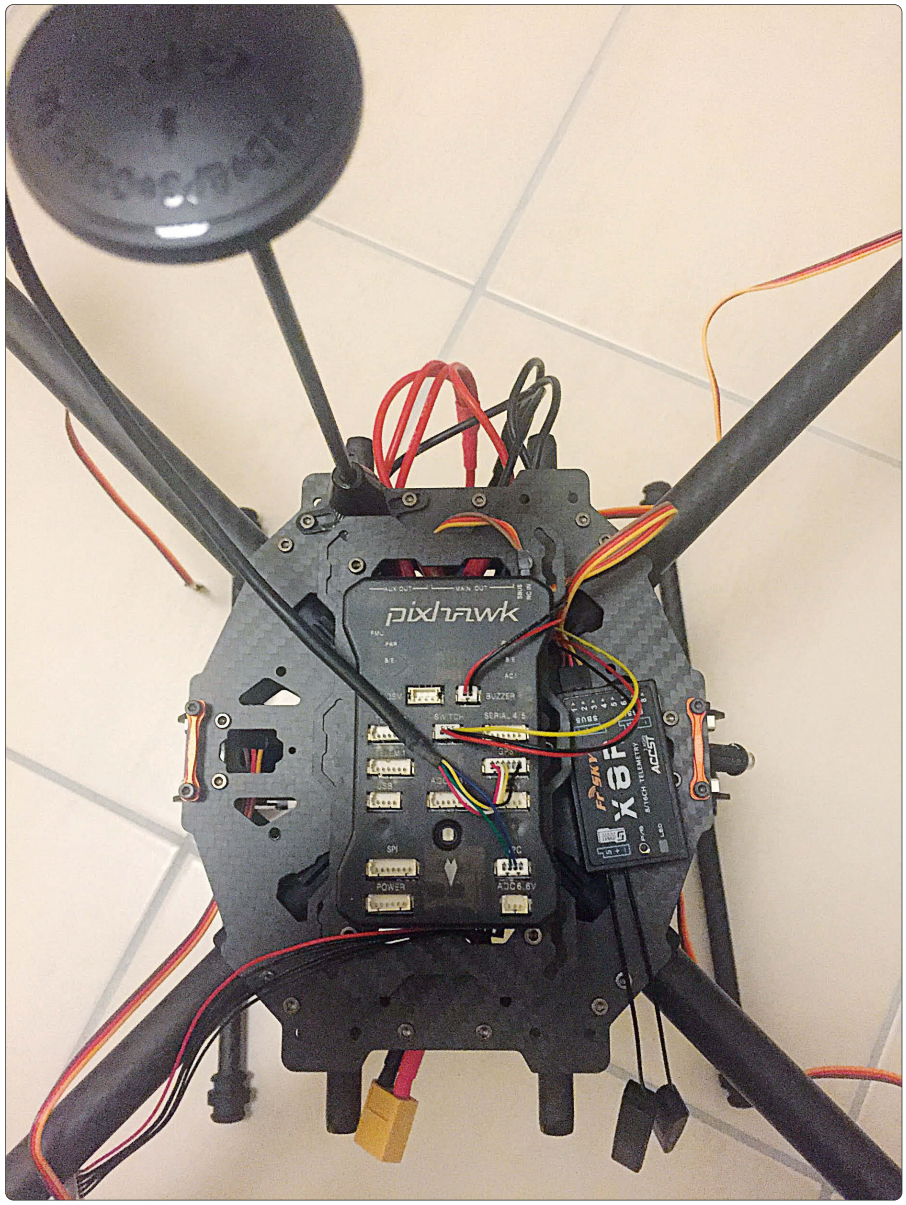
(les nouvelles fonctionnalités viennent donc au bon gré du constructeur).

- **APM :** un contrôleur à base Arduino, développé par la communauté ArduPilot. C'est l'un des précurseurs des contrôleurs de vol et il est idéal pour les drones voués à la prise de vue. La technologie embarquée date un peu, mais il reste accessible aux bourses modestes (40 - 100 €). Le logiciel est open source donc on peut modifier son fonctionnement à son gré. L'interface avec le contrôleur est simple, car ce dernier prend en charge le protocole de communication MAVlink.
- **Pixhawk :** un concurrent direct du DJI Naza, fabriqué par la société 3D Robotics. C'est une version améliorée de l'APM car il utilise le même code source pour fonctionner, mais avec des composants électroniques plus récents et plus puissants. Bien qu'assez cher (170 - 250 €), il peut néanmoins réaliser l'ensemble des tâches que l'on peut attendre d'un contrôleur, avec l'avantage lui aussi d'être basé sur un logiciel open source et d'offrir la possibilité d'ajouter des fonctionnalités. On peut également s'interfacer au contrôleur via le protocole MAVlink.

Pour la réalisation de mon propre drone, mon choix s'est porté vers le contrôleur Pixhawk de 3D Robotics et ce pour plusieurs raisons :

- Il est parfaitement adapté pour les prises de vues ;
- L'alimentation du contrôleur est simple, via un adaptateur fourni que l'on branche directement sur la batterie. C'est donc beaucoup moins compliqué que d'utiliser le BEC (pour Battery Eliminator Circuit) des ESC mais en contrepartie, on est limité dans le choix des batteries car l'adaptateur ne supporte pas tous les voltages ;
- Il est compatible avec le protocole de communication MAVLink et il existe une API pour pouvoir contrôler le drone via la programmation (dans mon cas, un programme présent sur un Rapsberry Pi) ;
- Il offre une grande variété de modes comme le maintien à une position, la programmation de parcours, le retour à la position de départ, le « failsafe » en cas de perte de communication ;
- Il m'a été recommandé par l'ensemble des professionnels du drone que j'ai pu rencontrer pour la prise de vue.

Le montage du contrôleur doit se faire sur un dispositif anti-vibration de façon à ne pas trop perturber l'accéléromètre et il faut bien réfléchir à son emplacement, si possible assez éloigné de la batterie pour éviter toute perturbation électrique. Lors de la première utilisation, il faut uploader le firmware du contrôleur, ainsi que réaliser l'ensemble des calibrations pour que le contrôleur soit opérationnel. On réalise ces opérations grâce aux logiciels Mission Planner (Windows) ou APM Planner (MAC, linux). Il existe des dizaines de vidéos sur Youtube, dont certaines très bien faites, qui expliquent comment réaliser l'upload et la première configuration du contrôleur.



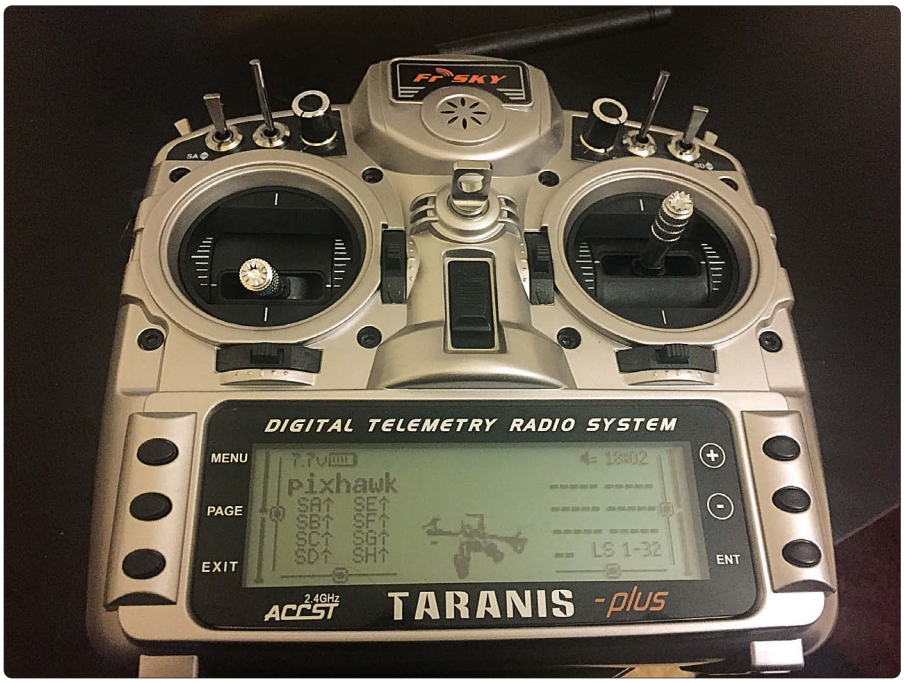
Dans le cas de mon propre drone, j'ai associé mon contrôleur de vol à un module GPS compatible avec le pixhawk, de façon à lui ajouter toutes les fonctionnalités liées au GPS : programmation de circuit, maintien de la position, retour au point de départ...

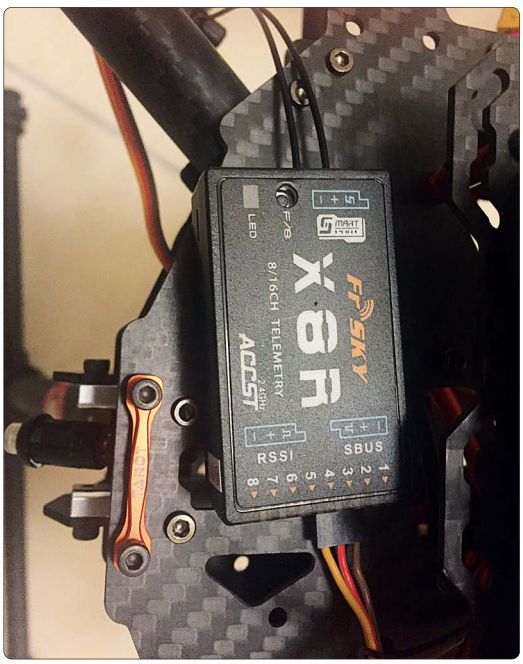
Radio commande + Récepteur : un duo pour garder les commandes

Il va sans dire que le plaisir de faire voler un appareil tel qu'un drone tient pour beaucoup dans le fait de pouvoir le contrôler et qu'il nous obéisse au doigt et à l'œil. L'instrument de tout pilote qui se respecte en modélisme est bien entendu la radiocommande. Comme pour chacune des pièces présentées jusqu'ici, il existe une large gamme de radiocommandes pour toutes les bourses. Si cet élément est l'un des plus chers de votre processus de fabrication, il est également le plus facilement réutilisable pour d'autres projets.

Les radiocommandes sont plus ou moins complexes et en plus des deux joysticks qui permettent de faire évoluer votre appareil dans toutes les directions, vous allez trouver sur le marché des radios commandes avec toute une panoplie d'interrupteurs et de boutons, qui dans le cas d'un drone, ne vous serviront pas nécessairement.

Autre point de vigilance, vérifiez bien lorsque vous achetez votre radiocommande que son





mode correspond à vos habitudes. En effet, il y a deux modes possibles pour une radiocommande : le mode 1 qui correspond à la manette des gaz à droite et le mode 2 qui est avec la manette des gaz à gauche. Ce choix ne peut être modifié de façon logicielle car la manette des gaz se base sur un mécanisme particulier, donc pensez-y en amont !

La majorité des radiocommandes ont au moins 8 canaux de communication - un canal correspondant à un élément de votre radiocommande (par exemple interrupteur, bouton, axe d'un joystick). Pour vous donner un exemple, le joystick qui gère le roulis1 et le tangage2 du drone utilise deux canaux sur ma radio : un canal qui va transmettre la valeur du roulis, et un autre qui va transmettre la valeur du tangage. De base, notre radiocommande a besoin de 5 canaux : roulis(1), tangage(2), lacet, gaz, sélection du mode. Rien n'empêche par la suite d'utiliser des canaux supplémentaires pour diriger la nacelle où se trouve la caméra par exemple.

En résumé, un canal correspond à un élément à contrôler. Mais pour que les canaux ouverts sur la radiocommande impactent le vol de notre drone, la transmission au contrôleur de vol pour interprétation est nécessaire. C'est là que le récepteur entre en jeu, car il va servir de pont entre les deux.

Le récepteur communique avec la radiocommande via des ondes radios, puis il les transmet au contrôleur de vol via un ou des branchements. Il existe en effet différentes façons de connecter votre récepteur à votre contrôleur. Schématiquement, voici l'alternative :

- Soit une sortie par canal de communication, que vous branchez sur une entrée correspondante du contrôleur,
- Soit un branchement unique via un port "sbus" sur votre récepteur et votre contrôleur.

L'avantage de la deuxième solution (branchement unique) est qu'en plus de n'avoir qu'un câble à connecter, ce mode vous permettra sur certains récepteurs d'utiliser jusqu'à 16 canaux, là où la première méthode vous limitera à 8 canaux, car les récepteurs sont munis uniquement de 8 ports pour 8 sorties. Sachez également que certains contrôleurs de vol ne peuvent recevoir les données du récepteur que via leur port "sbus". Dans ce cas, si votre récepteur n'est pas compatible, il existe un adaptateur (le PPM Sum Receiver), qui pourra faire l'intermédiaire entre votre récepteur et votre contrôleur.

Pour ma part, j'ai choisi d'utiliser une radiocommande de la marque FrSky : la Taranis 9D plus. C'est une radiocommande de milieu de gamme, animée par un logiciel open source OpenTx désormais réputé dans le monde des multicopters car il offre beaucoup de possibilités de personnalisation. Je l'ai associée avec le récepteur XR8 de la même marque. Il présente l'avantage d'avoir un port "sbus", et donc mon branchement n'en sera que simplifié. Il faudra là aussi passer par le logiciel de configuration du contrôleur pour calibrer la radiocommande. Attention, lorsque vous choisissez un récepteur, vérifiez auparavant sur Internet la compatibilité avec votre contrôleur, car il se peut que certains modèles ne soient pas complètement compatibles entre eux!

Un dernier point à anticiper concernant le récepteur : le positionnement des antennes. Ce point est essentiel pour avoir une bonne réception lorsque le drone est en l'air, ce n'est donc pas un détail ! Les récepteurs sont dotés de deux antennes et il est recommandé de les orienter dans des sens différents pour augmenter la qualité de réception. Certains diront de les positionner au-dessus du drone, d'autres en dessous, je pense pour ma part que le positionnement dépend vraiment de la forme de votre drone et de la forme des antennes, et qu'il vous faudra réaliser des tests dans plusieurs configurations pour optimiser votre réception. Avec l'avènement des imprimantes 3D, vous pourrez trouver très facilement des modèles de support d'antenne correspondant à votre récepteur, à télécharger et à imprimer.

Avec ces nouveaux éléments (contrôleur, radiocommande et récepteur), mon drone est maintenant très avancé. Je n'ai pas eu besoin de fer à souder pour cette partie, mais j'ai dû user de scotch double face et d'un collier de serrage pour positionner les éléments liés au contrôleur de vol et au récepteur. Il me reste à choisir une batterie et je pourrai faire mes premiers tests de vol, vérifier que le GPS fonctionne correctement et que mes antennes sont bien positionnées pour avoir une bonne réception.

La suite au prochain épisode...

(1) Roulis (roll) : mouvement de rotation autour de son axe longitudinal. Le drone se penche vers la droite ou vers la gauche.

(2) Tangage (pitch) : mouvement de rotation autour de son axe transversal. Le drone se penche vers l'avant ou vers l'arrière.

MongoDB, la base de données NoSQL qui implémente le modèle orienté document bouscule depuis une dizaine d'années les systèmes de gestion de bases de données classiques.

Ce nouvel article sur les bases NoSQL suit le même ordre d'idées que l'article sur le modèle orienté graphe paru dans le numéro 188 de Programmez. Mon objectif est de présenter une évaluation du modèle orienté document à l'aide d'un ensemble de critères prédéfinis.

Fabrice Chapuis
Consultant informatique, spécialiste en bases de données
fchapuis@ip-worldcom.ch

Cet article n'a pas pour but de donner des explications sur l'installation et la configuration du logiciel MongoDB. Des commandes utilisées avec ce logiciel sont toutefois proposées à titre d'exemples dans les rubriques d'évaluation des critères. Ces critères ainsi que les mots en **gras** dans le texte sont expliqués dans le glossaire technique.

Le modèle orienté document

Le modèle orienté document est représenté dans la Fig.1. Dans ce modèle une clé est associée à un document, ce document est constitué d'un ensemble de champs imbriqués. Chacun de ces champs peut être indexé individuellement. Les documents sont regroupés au sein d'une collection. Une base de données orientée document contient un ensemble de collections, ces collections seront matérialisées lorsque des données seront ajoutées dans la base. Les logiciels les plus répandus implémentant le modèle orienté document sont CouchDB et MongoDB. La structure imbriquée du modèle orienté document se rapproche de celle du XML. Cette structure a l'avantage d'être compatible avec les systèmes à objets.

Fig.1

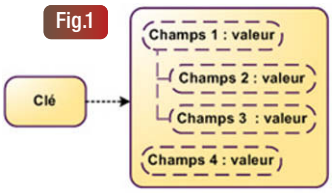


Figure 1: Le modèle orienté document

MongoDB a été choisi comme logiciel de référence pour notre étude. C'est la compagnie 10gen qui développe MongoDB depuis 2007. La dernière version disponible est la version 3.0, vous pouvez la télécharger sur le site www.mongodb.com. Ce logiciel est très populaire, de nombreuses compagnies telles que ebay, Forbes, Bosch, Doodle, le CERN ont implémenté MongoDB. Les données sont stockées sous la forme d'objets binaires dans des fichiers de type **BSON**. La communication entre l'utilisateur et la base se fait à l'aide d'un SHELL Javascript.

Critères d'évaluation du modèle:

La liste de critères ci-dessous sera utilisée pour évaluer le modèle orienté document.

- Nature des données (**structurées**, **faiblement structurées**);
- Relation entre les données (**intégrité référentielle**, **relations hiérarchiques**);
- Cycle de vie des données (**versioning**, **Time To Live** (TTL));
- **Schémas** et opérations **CRUD;**
- Consistance des données (**propriétés ACID**);
- Performance (**indexation**, **partitionnement**);
- Volumétrie (**Big Data**);
- Analyse des données (**agrégation des données**);
- Persistance et tolérance aux pannes (**réplication);**
- Confidentialité des données (**droits d'accès**).

Nature des données

Données structurées [1]

Les bases de données orientées documents telles que MongoDB stockent des documents au format JSON. JSON signifie JavaScript Object Notation, c'est un format de définition de données qui contient une structure hiérarchique de propriétés et de valeurs. C'est un standard ouvert, humainement lisible. Avec le XML, JSON est le format principal pour l'échange de données utilisé sur le Web moderne. Comparativement à XML, JSON est plus compact. JSON prend en charge tous les types de données de base: les nombres, les chaînes et les valeurs booléennes, ainsi que des tableaux. L'exemple suivant représente un document au format JSON :

```
{"firstName": "Bidhan",
"lastName": "Chatterjee",
"age": 40,
"address":{"streetAddress": "144 J B Hazra Road", "city": "Burdwan", "state": "Paschimbanga",
"postalCode": "713102"},
"phoneNumber":[{"type": "personal","number": "09832209761"},{"type": "fax","number":
"91-342-2567692"}]}
```

Données faiblement structurées [2]

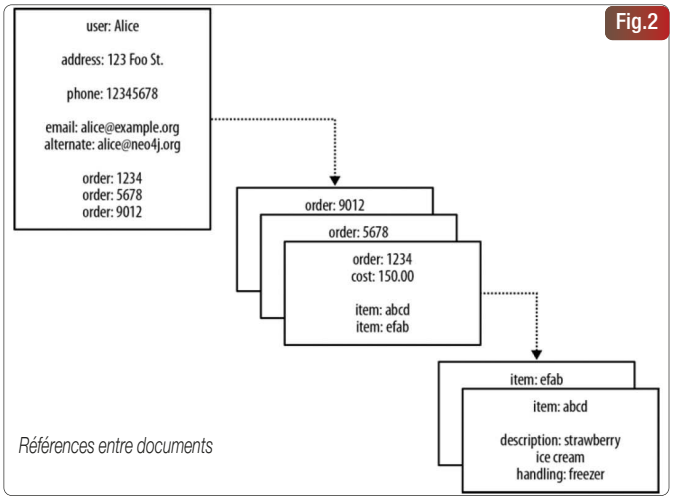
Le format JSON remplit le même cahier des charges que le XML, en tout cas sur le principe, il est donc taillé pour stocker des données semi-structurées. Des documents contenant du texte libre peuvent également être stockés sous forme de paires clé-valeur, un index configuré sur ce champ permet de rechercher une chaîne de caractères à l'intérieur du texte.

Relation

Intégrité référentielle [3]

Le modèle orienté document est fortement dénormalisé, un document contient toute les informations liées à une entité. Pour pouvoir réaliser des relations dans ce modèle, il faut ajouter l'identifiant d'un document dans un

Fig.2



Références entre documents

autre document, ce qui revient à créer une « **clé étrangère** ». C'est l'application qui doit gérer ces identifiants pour pouvoir mettre les documents en relation. Dans l'exemple proposé dans la Fig.2, vous pouvez connaître les produits qui ont été achetés par Alice, en revanche la réciproque est plus complexe, si l'on souhaite connaître qui a acheté une crème glacée, il est nécessaire de parcourir tous les utilisateurs ou créer des « back références ». L'application doit gérer également les références lorsque des documents sont supprimés.

Relations hiérarchiques [4]

De par sa nature un document JSON est bien adapté pour structurer des données hiérarchiques.

Cycle de vie

Versioning [5]

Dans le modèle relationnel si la structure d'une table doit être modifiée dans le but de faire évoluer une application et qu'il est nécessaire de conserver l'ancienne structure, la table d'origine devra être renommée afin que le nom de cette table contienne un numéro de version. Dans le modèle orienté document, grâce à la souplesse de sa structure, c'est beaucoup plus simple, il suffit de rajouter un champ dans le document pour le numéro de version.

TTL [6]

Il est possible d'associer une durée de vie à un document. Un « Garbage Collector (GC) » va se charger de supprimer les documents à l'échéance du délai. Ce GC est exécuté toutes les 60 secondes. Cela signifie que certains documents peuvent exister encore quelques secondes après avoir expiré. Pour configurer un TTL, il faut créer un index sur un champ de type « date ». La commande suivante permet de supprimer les données de session d'un utilisateur après une heure.

```
db.sessions.ensureIndex( { "timestamp": 1 }, { expireAfterSeconds: 3600 } )
```

Lorsque le temps est expiré, c'est tout le document qui est supprimé. Il est également possible de configurer une date à laquelle le document sera éliminé.

Schéma et opérations CRUD

Schéma [7]

Le modèle orienté document permet de manipuler des objets sans schéma prédéterminé. Dans Mongo, il n'y a pas de tables ni de colonnes et on ne spécifie pas de type pour les données. Les documents sont stockés dans des collections. Par analogie avec le modèle relationnel, les collections sont l'équivalent des tables et les documents l'équivalent des tuples. Les documents stockés dans une collection peuvent avoir un nombre de champs qui varie. En règle générale, les documents seront stockés avec la même structure, mais la souplesse du schéma est un des avantages de ce modèle.

Opérations CRUD [8]

La manipulation des données contenues dans une collection se fait à l'aide de fonctions. La syntaxe utilisée est la suivante:

```
db.NomDeCollection.NomDeFonction()
```

CREATE :

Pour insérer une donnée, on utilise la fonction insert()

```
db.sessions.insert({id: 1234, nom: 'Lorentz', prenom: 'Bill', timestamp : new Date()})
```

Pour chaque document inséré dans une collection, MongoDB génère un identifiant unique. Cet identifiant joue le rôle de clé primaire, il permet d'identifier de façon unique un document dans une collection.

READ:

Pour récupérer tous les documents d'une collection on utilise la fonction find()

```
db.sessions.find()
```

UPDATE:

Les mises à jour sur un document peuvent s'effectuer de deux manières :

- Un document peut être remplacé par un nouveau document

```
db.sessions.update({nom: 'Lorentz'}, { languages: ['ruby','c','c++']})
```

- L'opérateur $set permet de faire une mise à jour partielle du document sans écraser le document d'origine.

```
db.sessions.update({nom: 'Lorentz'}, {$set: {'taille': 184} })
```

DELETE:

Pour supprimer un document on utilise la fonction remove().

```
db.sessions.remove({nom: 'Lorentz'})
```

En comparaison des autres SGBD NoSQL, MongoDB propose de très nombreuses fonctionnalités pour manipuler des données contenues dans les documents, cependant on est encore très loin des possibilités qu'offre le langage SQL.

Consistance des données :

Propriétés ACID [9]

MongoDB est une base « ACID non compliant », c'est à dire que les verrous traditionnels et les transactions impliquant plusieurs documents ne sont pas supportés. En revanche elle permet d'effectuer des opérations atomiques sur un document. Par exemple, retirer un élément du stock pour l'ajouter aux ventes dans une seule et même transaction n'est pas possible, à moins que stock et vente ne se trouvent dans le même document. Pour pallier ce problème on peut imaginer d'utiliser MongoDB en conjonction avec une base qui supporte les transactions, comme Postgress par exemple. Il s'agit dans ce cas d'une approche multi-modèles.

Performance

MongoDB n'a pas de cache configurable, la base utilise automatiquement toute la mémoire à disposition sur la machine par l'intermédiaire de fichiers « map » en mémoire. A la différence des SGBDR qui transforment les données en représentations objets utilisables par les applications, la représentation d'un document en mémoire dans Mongo est la même que celle qui se trouve sur le disque; cette mise en cache est moins coûteuse en terme de ressources. La base garde en mémoire toutes les données récemment utilisées qui peuvent tenir dans la RAM.

Indexation [10]

Par défaut toutes les collections ont un index sur le champ d'identification du document (cet identifiant technique est codé sur 12 bytes). Il est possible d'ajouter des index secondaires sur n'importe quel champ apparaissant dans le document. Dans la Fig.3, le champ « category » a été indexé pour permettre par exemple d'accélérer une recherche pour retrouver les documents qui ont comme valeur « networking » .

La Fig.4 présente la structure d'un index dans MongoDB. Un index est constitué d'une copie du champ à indexer (le champ *nom)* associé à l'identifiant du document (la clé). Ces index sont appelés index simples. Il existe d'autres types d'index dans MongoDB: comme par exemple les index composés, les index multi-clés. Un index composé possède des références sur plusieurs champs appartenant à un document. Les index multi-clés permettent d'indexer les éléments d'un tableau.

Partitionnement [11]

MongoDB est construit pour l'évolutivité applicative et la haute disponibilité. Un cluster mongo est composé d'un ensemble de noeuds possédant chacun leur propre jeu de données. Les données insérées sont automatiquement distribuées sur l'ensemble de ces nœuds de façon à équilibrer la charge. La Fig.5 montre un exemple d'architecture distribuée MongoDB.
Le *Config Server* contient toutes les informations liées à la topographie du « cluster ». *Mongos* est le processus coordinateur; ce « **deamon** » attend les connexions des applications et distribue les écritures sur les noeuds du cluster. Les partitions (« shard » en anglais) sont les serveurs membres. Etant donné que les données sont répliquées sur plusieurs noeuds, les requêtes sont distribuées sur un ensemble de serveurs et par conséquent garantissent un temps de réponse optimal. La distribution des données permet de répondre à une augmentation significative du volume des données à traiter.

Volumétrie

Big Data [12]

L'explosion de l'utilisation des téléphones portables, la multiplication des différents appareillages de recueil d'informations (stations météo, télescope, séquenceurs d'ADN, données de flux RSS) produisent des volumes de données gigantesques (mégadonnées). De nombreuses entreprises utilisent MongoDB pour ses capacités à stocker de grands volumes d'informations.

Analyse

Agrégation des données [13]

MongoDB implémente l'algorithme Map-Reduce. Cet algorithme effectue des opérations d'agrégations sur des collections. Prenons un exemple pour expliquer son fonctionnement. Le Tableau ci-dessous représente la collection *ordres*. Cette collection contient l'ensemble des produits vendus par une compagnie. Chaque ligne correspond à un document Mongo. On veut calculer le montant total pour chaque produit.

no_comm	id_client	id_article	quantite	prix
25	AA	X	3	20
34	CC	Y	2	40
44	CC	Z	1	10
36	AA	Y	5	40
40	BB	Z	7	11
44	CC	X	2	20
48	AA	Y	1	40

La fonction Map prend en paramètre un couple (clé-valeur), la clé correspond à l'identifiant des articles, la valeur est une opération (prix X quantite)

```
var mapf = function() {
   emit(this.id_article, this.quantite*this.prix);
}
```

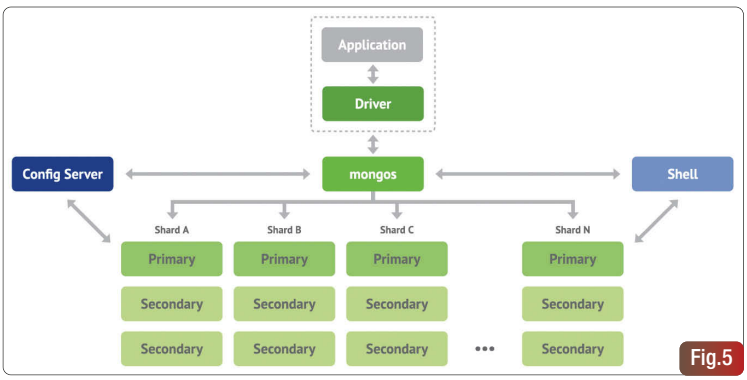


Fig.5

Cluster MongoDB

Les 3 tables ci-dessous montrent le résultat obtenu avec la fonction map, les données sont triées par produit.

id_article	total
X	60
X	40

id_article	total
Y	80
Y	200
Y	40

id_article	total
Z	87

La fonction Reduce prend en paramètre l'id des articles et le champ à agréger.

```
var reducef = function(id_article, montant){
      return Array.sum(montant);
};
```

Le résultat obtenu est affiché dans les 3 tableaux suivants :

id_article	total
X	100

id_article	total
Y	320

id_article	total
Z	87

La commande suivante va déclencher les calculs.

```
db.ordres.mapReduce(
  mapf,
  reducef,
  {out: "map_reduce_commandes"}
)
```

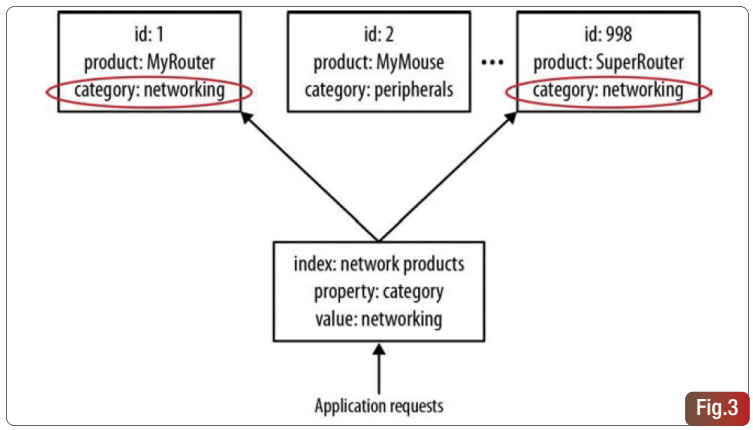


Fig.3

Indexation dans le modèle orienté documents

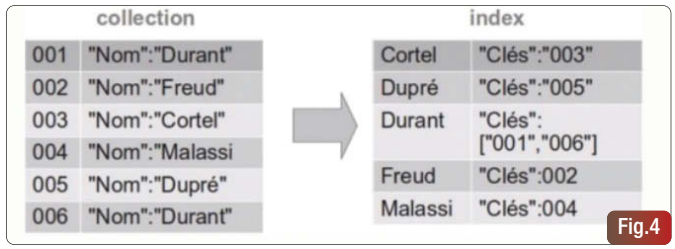


Fig.4

Structure d'un index

La sortie est renvoyée dans une collection. Pour obtenir le résultat, il suffit d'afficher le contenu de la collection *map_reduce_commandes*.

Persistance et tolérance aux pannes

Si une base Mongo n'est pas arrêtée correctement, suite à un défaut d'alimentation par exemple, alors cette base se retrouve dans un état inconsistant, c'est à dire que les dernières écritures sur la base n'ont pas encore été enregistrées dans les documents. Dans mongo, les écritures sont poussées dans un fichier journal toutes les 100 millisecondes. Cela signifie que les écritures sont persistantes et qu'une instance Mongo est capable d'effectuer une reprise sur panne (instance crash recovery).

Réplication [14]

La réplication maître-esclave est le mode de réplication supporté par MongoDB. Ce mode est flexible et permet d'augmenter la disponibilité du système. Il peut en outre être utilisé pour faire de la reprise sur panne automatique. Comme il existe plusieurs copies des données, les réplicas protègent le système lors de la perte d'un serveur.

Le principe de fonctionnement de la réplication dans MongDB est le suivant : une instance primaire (processus mongod) reçoit toutes les opérations en écriture, les données modifiées sont transférées dans un fichier journal (oplog), les instances secondaires vont mettre à jour leur jeu de données à partir de ce fichier journal. Dans le cas où l'instance primaire tombe, un des membres va prendre le relai à la suite d'une élection. Il est possible de configurer un arbitre afin de faciliter ce vote. L'élection d'un nouveau membre pour prendre la fonction d'instance primaire est réalisée de façon automatique (failover automatique).

Confidentialité des données

Droits d'accès [15]

Par défaut l'authentification avec mot de passe n'est pas activée sur MongoDB, il est possible de l'activer au démarrage du serveur.

Il existe par défaut une base *admin* dans MongoDB. Un utilisateur administrateur possède des privilèges supplémentaires par rapport aux autres utilisateurs, comme celui par exemple d'arrêter le serveur de bases de données. Les droits d'accès aux ressources de la base sont gérés grâce à des rôles. Les utilisateurs peuvent avoir plusieurs rôles. Il n'existe actuellement aucune option pour chiffrer des documents dans MongoDB.

Conclusion

La matrice ci-dessous est une synthèse de l'ensemble des critères étudiés pour le modèle orienté documents. Malgré le fait que les transactions dans ce modèle ne sont pas prises en charge par le moteur de bases de données, en effet c'est l'application qui prend à son compte la gestion de l'intégrité des données, ce modèle a l'avantage d'être beaucoup plus souple que le modèle tabulaire que l'on retrouve dans le monde relationnel. Les clusters MongoDB offrent de grandes possibilités en termes de distributions des données et de haute disponibilité. C'est peut-être pour cela, qu'actuellement, MongDB est une des bases de données NoSQL qui rencontre le plus de succès.

Glossaire technique

ACID (Atomicité, Consistance, Isolation, Durabilité) (Harder and Reuter, 1983) : c'est un ensemble de propriétés qui garantissent que les transactions dans la base de données sont fiables.

BSON: Binary JSON

Clé étrangère: une clé étrangère est une colonne d'une table qui référence une colonne d'une autre table (clé primaire). Une clé étrangère sert à garantir l'intégrité référentielle des données.

Rubriques	Catégories		Modèle orienté graphe
Données	Nature des données	1	Données structurées
		2	Données faiblement structurées
	Relation entre les données	3	Intégrité référentielle
		4	Relations hiérarchiques
	Cyle de vie	5	Versionning
	Cyle de vie	6	TTL
Requêtes	Schéma et opérations CRUD	7	Schéma
		8	Opérations CRUD
	Consistance des données	9	Propriétés ACID
Performance	Peformance	10	Index
		11	Partitionnement
	Volumétrie	12	Big Data
Analyse	Analyse	13	Agrégation des données
Persistance	Tolérance aux pannes	14	Réplication
Sécurité	Confidentialité	15	Droits d'accès, encryption

mauvais (dégradation importante ou pas implémenté)
possible
bon, point clé du modèle

CRUD : Create, Read, Update, Delete

Dead Lock : un « Dead Lock » (en français verrou mortel) se produit lorsque deux transactions se bloquent mutuellement.

Dirty Read : un « Dirty Read » se produit lorsqu'une transaction lit une donnée modifiée par une autre transaction sans que la modification ait été validée.

Données structurées (Abiteboul et al.,1999) : des données structurées suivent un schéma prédéfini.
Les données vont se conformer aux spécifications fournies dans le schéma. Exemple : base de données relationnelles.

Données faiblement structurées : ce terme signifie qu'il n'y a aucune structure identifiable pour ces données. Les données non structurées sont aussi décrites comme des données ne pouvant pas être stockées dans des tables (un document dans un répertoire, une vidéo, une image).

Index : structure organisée contenant des données (arbres) permettant d'accélérer la recherche dans une base de données.

Intégrité référentielle : les contraintes d'intégrité référentielle permettent au SGBD de gérer automatiquement la présence de données référencées dans différentes relations de la base. La notion de lien permet de spécifier de telles propriétés.

Relation de hiérarchie : les relations de hiérarchie sont représentées par des arbres. Les relations entre les nœuds de cet arbre sont de type parent-enfant.

Réplication: la réplication est un processus qui consiste à maintenir plusieurs copies d'une même base de données.

Schéma de données: il permet de décrire formellement la structure des données, leurs types. On parle de tables dans le cas des bases de données relationnelles.

Versioning : le « versioning » est généralement associé à une stratégie de migration de données qui a pour but de déplacer les données d'une ancienne version vers une nouvelle

Définir notre orientation en Drupal

Après 5 ans de développement, le 19 novembre 2015 sortait Drupal 8. Il est important de penser aux utilisateurs et à nos lecteurs, car de nombreux projets utilisent toujours des versions antérieures comme Drupal 5 ou 6. Mais alors que faut-il faire réellement ? Drupal 7 ou Drupal 8 ?

Christophe Villeneuve
Consultant IT pour Neuros, Mozilla Reps, auteur du livre "Drupal avancé" aux éditions Eyrolles et auteur aux Éditions ENI, Rédacteur pour WebRIVER, membre des Teams DrupalFR, AFUP, LeMug.fr (MySQL/MariaDB User Group FR), Drupagora...

Quelques rappels

Maintenant que Drupal 8 est disponible, la version 6 de Drupal est passée en version archivée, c'est-à-dire que cette version ne sera plus maintenue sauf pour les failles de sécurité. Cette version passe dans les mains de l'équipe Core mainteneur. Mais il est préférable d'effectuer la migration vers une version plus récente. Du côté de la version 7, elle sera maintenue jusqu'à la version 9 du CMS dont la date de sortie n'est pas encore définie, et la roadmap pas encore déterminée. Cette version a mis 14 mois à s'imposer dans l'univers du CMS, car les modules les plus utilisés n'étaient pas tous portés. Même si la version 8 est disponible, tous les modules indispensables ne sont pas complètement portés, ce qui ralentira votre migration.

L'article va se positionner en mode intermédiaire, car il est facile de migrer de Drupal 5 ou 6 vers Drupal 7 ou de maintenir cette version, en attendant que vos équipes puissent développer sur Drupal 8. Bien entendu, il ne faut pas louper le train, c'est pourquoi vous pouvez commencer à réaliser quelques petits projets pour bien démarrer (voir Programmez 192 de janvier 2016).

Fig.1

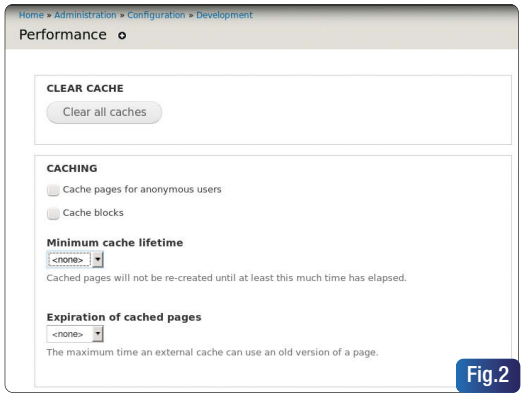


Fig.2

Le mode avancé

Le mode avancé est un mode intermédiaire entre 2 versions majeures, car vous vous trouvez dans une position entre 2 chaises. D'un côté vous ne possédez pas encore les moyens et les ressources pour effectuer la migration vers Drupal 8, de l'autre les utilisateurs réclament des améliorations, car c'est leur outil de travail. Toutefois, la technologie Web évolue tous les jours et vous devez pouvoir répondre aussi bien sur les problèmes de performances, de sécurité, tout en ajoutant de nouvelles fonctionnalités. L'article va lister un certain nombre de points qu'il est important de prendre en compte pour que votre application reste toujours appréciée des utilisateurs.

Performance

Cette notion se positionne toujours en premier point et fonctionne toujours en parallèle lors de la maintenance, c'est pourquoi la notion de cache est importante. Il en existe de différentes formes : tout d'abord, le cache interne passe par des extensions du langage comme Alternative PHP Cache (APC) ou du Memcache. Ensuite, le cache externe est intéressant quand votre projet doit répondre à un fort trafic. Vous trouverez par exemple Varnish ou Akamai qui sont complètement intégrés à Drupal sous la forme de modules. Toutefois, il ne faut pas en abuser, car souvent vos problèmes viennent d'une mauvaise configuration du serveur ou des modules mal utilisés ou abandonnés en cours de route. Fig.1.

Maintenance

Quand vous ne bénéficiez pas de TMA (Tierce Maintenance Applicative), c'est que vous êtes le seul à maintenir votre projet internet. Celle-ci ne se limite pas à mettre à jour les modules que vous utilisez depuis le début du projet. Il faut aussi répondre aux attentes des utilisateurs et des éventuelles évolutions ou de configurations des utilisateurs.

Il est très facile d'effectuer des erreurs lors de la maintenance d'un projet, qui vous feront perdre de nombreuses heures de travail pour résoudre une case mal cochée ou la construction d'une mauvaise vue.

Les erreurs rencontrées sont souvent liées à une mauvaise connaissance du CMS. Les effets non désirés sont provoqués quand vous n'utilisez pas la couche API de Drupal, c'est à dire, qu'il existe des fonctions prédéfinies pour Drupal que vous devez utiliser. Elles sont disponibles à l'adresse suivante : https://api.drupal.org/api/drupal .

Une autre erreur concerne l'utilisation des requêtes globales, qui sont très pratiques, mais souvent inutiles, car elles sont très gourmandes au niveau des ressources de la machine.

De plus, l'autre critère souvent oublié est la volumétrie des données qui est souvent sous-estimée. Enfin, il n'est pas rare de retrouver du code PHP dans le thème ou le template alors qu'il existe des emplacements bien définis.

Bien entendu, il existe aussi des moyens d'optimiser vos fichiers CSS, JavaScript, d'exécuter des CRONs et d'éviter les pages 404. Tout ceci peut être manipulé à partir de l'interface. Fig.2.

Toutefois, vous pouvez résoudre l'ensemble de ces soucis avec les nombreux modules et outils du marché pour corriger et identifier les problèmes, comme Devel, New Relic, Xprof, BackFire... qui vous seront utiles.

L'environnement

Un environnement fait tourner votre projet Drupal. Il s'appuie souvent sur un socle AMP (Apache, MySQL, PHP). Cependant, il est indispensable d'en installer un dans votre ordinateur. Cette opération permet de tester de nouveaux modules, de concevoir et préparer les différentes évolutions des utilisateurs. La configuration AMP est disponible sur tous les systèmes d'exploitation (Linux, Windows, Mac), de cette façon vous êtes prêt à faire des essais sans risques. Ainsi, vous pouvez utiliser Wamp, Xampp, Mamp. De même, quand une évolution est prête, vous pou-

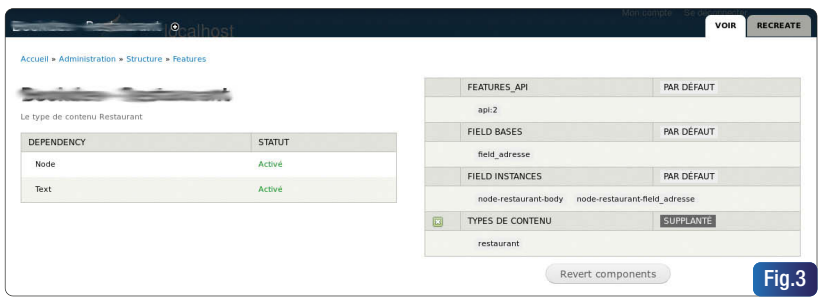


Fig.3

vez utiliser le module 'Features' (https://www.drupal.org/project/features) pour éviter de recommencer la configuration demandée. Fig.3.

Le back-end

La notion de contenu est une approche 'utilisateur'. Il en existe différentes sortes.

Tout d'abord les utilisateurs, qui ont la charge de fournir du contenu, tout en illustrant leurs articles par des images ou de la vidéo. Ensuite une autre catégorie de personnes peut animer, mettre en avant les contenus et donnera envie aux internautes de revenir vers votre site Internet ou votre application. L'ensemble des utilisateurs aura des droits, des rôles et des accès, qui pourront être plus ou moins complexes suivant la ou les compétences et leurs attributions.

Les solutions existent pour mettre tout cela en place, comme la gestion des droits, des rôles, etc. Il existe par exemple Organic Groups (https://www.drupal.org/project/og) Fig.4.

Bien sûr, la notion de groupe est facilitée si les modules sont déjà en place, car vous pouvez donner à un utilisateur des accès supplémentaires en cochant seulement des cases.

L'autre point aussi à cadrer concerne les TAGs car il n'est pas nécessaire de tout mettre en Tags. Ceci permet actuellement d'identifier du contenu plus facilement dans les moteurs de recherches et par la même occasion de mettre en avant les mots clefs les plus utilisés.

Le front-end

Le front-end est la partie visible de votre projet Web. Les internautes vont consulter vos différentes pages et informations avec les différents devices. Ce point évolue peu, car il est associé aux squelettes de vos pages. Cependant les vues et les animations seront des approches à connaître, car la mise en avant d'un bloc est associée à une partie d'un site Internet. Par contre les animations seront associées à un effet à la mode ou sur une période déterminée (les soldes par exemple).

Recherche

Quel que soit le volume d'informations, de pages, de contenus, vous aurez à résoudre les demandes de recherches. Même si la fonctionnalité de recherche est disponible dans le CMS, il faut souvent prévoir un complément ou utiliser des applications externes comme Solr (https://www.drupal.org/project/search_api_solr), elasticSearch (https://www. drupal.org/project/elasticsearch), Sphinx (https://www.drupal.org/project/sphinx)... Il est important de prévoir cette fonctionnalité en amont, néanmoins il existe toujours des solutions pour vous aider à améliorer ces fonctionnalités a posteriori. Fig.5.

Un nouveau type de contenu

L'ajout d'un nouveau type de contenu engendre de nombreuses opérations et un travail souvent mal quantifié si vous ne connaissez pas bien le CMS. Fig.6. Lorsque vous souhaitez un nouveau type de contenu, il passe obligatoirement par la création de celui-ci. Ensuite, les droits d'accès, les rôles doivent être clairement identifiés pour définir quels seront les utilisateurs autorisés à effectuer la saisie. Enfin, vous prévoyez aussi comment sera présentée la vue dans un bloc ou une page, avec ou sans animation... L'ensemble de ces évolutions peut vous amener à installer un nouveau module ou d'en concevoir un ou plusieurs modules spécifiques à votre métier. Chaque nouveauté doit faire appel à l'API de Drupal, car si vous réalisez l'opération à la volée, vous rencontrerez des effets de bords non voulus à l'origine.

Bien entendu, pour boucler cette évolution, il faut effectuer différents types de tests : unitaires, fonctionnels... car vous partez sur un site existant et vous ne pouvez pas vous permettre de tout casser lors de l'insertion de cette évolution.

Déploiement

Il s'agit aussi d'une étape importante comme le reste, car il va permettre à tous les utilisateurs de bénéficier des évolutions que vous avez apportées dans votre projet Drupal.

Pour cela, il est important de valider en amont auprès des demandeurs afin de pouvoir effectuer la livraison vers le serveur de production.

L'opération de déploiement peut se faire de différentes manières, dont nous aurons une préférence pour utiliser des logiciels de versioning.

La ligne de commande

Les lignes de commandes font peur, car il est plus facile d'utiliser l'interface et la souris pour effectuer des opérations. Bien sûr, certaines tâches ou opérations doivent passer par cette étape, mais pas la totalité.

C'est pourquoi il est important d'utiliser le mode Drush (https://www.drupal.org/project/drush) qui vous sera très utile pour gagner du temps.

Conclusion

Plusieurs cas se présentent à vous, si vous lancez un nouveau projet basé sur le CMS Drupal, il est préférable de commencer avec la version 8. Du côté de la migration, venant des anciennes versions, une migration en douceur peut être utile pour migrer vers la 7, voire la 8. Toutefois, si votre projet s'appuie sur la version 7, vous devez penser au coût de migration et le temps que cela prendra. Le choix n'est jamais évident, c'est pour cela qu'il existe de nombreuses sociétés qui sont là pour répondre à vos interrogations et attentes. Il est aussi possible de rentrer plus en profondeur par soi-même et de suivre les tutoriaux en ligne, ou d'acquérir le livre « Drupal avancé » aux éditions Eyrolles.

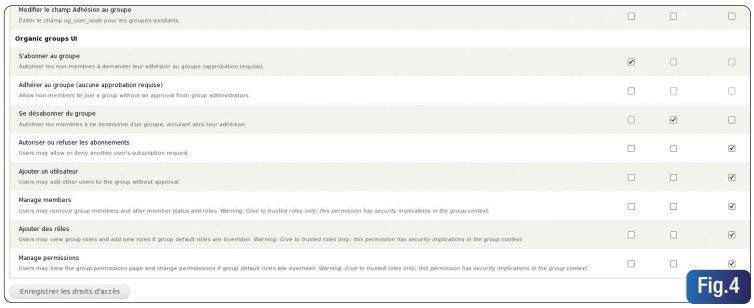


Fig.4

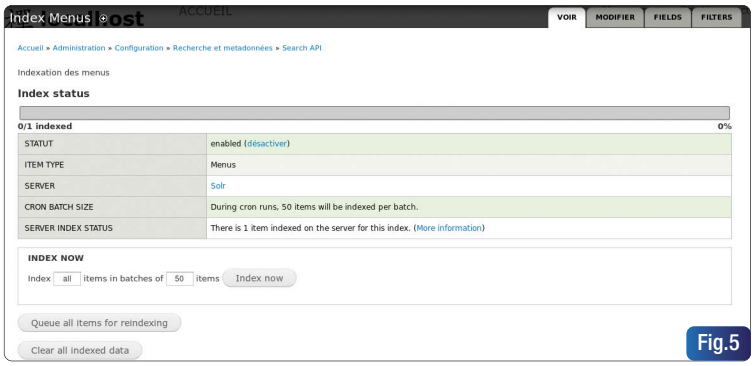


Fig.5

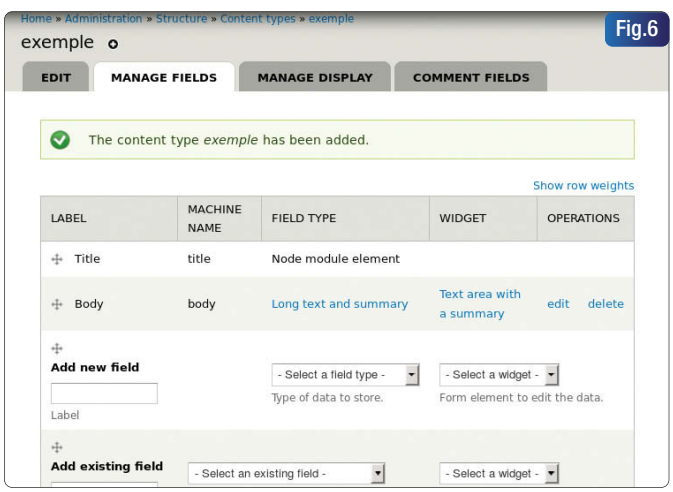


Fig.6

Timeline : 1985
Projet : Informix 4GL

Langage de programmation né en 1985, I-4GL est une déclinaison d'un langage de quatrième génération conçu originellement exclusivement pour les bases de données Informix. Aujourd'hui, il couvre de nombreuses plateformes et quasiment toutes les bases de données du marché, même NoSQL.

Jean Georges Perrin (jgp@jgp.net)
COO de 2CRSI Corporation, un constructeur de matériel informatique français. Auparavant, Jean Georges a travaillé chez Four J's, a fondé et dirigé plusieurs startups. Il est membre du directoire d'IIUG (International Informix Users Group, http://iiug.org) depuis 2002. Il vit aujourd'hui en Caroline du Nord.

Eric Vercelletto (eric.vercelletto@begooden-it.com)
Spécialiste des technologies Informix depuis 1986. Technical Account Manager chez Informix Software pendant plus de 10 ans, il y a co-réalisé le support de cours "Informix-4GL Advanced Programming". Fondateur de Begooden-IT Consulting. Il préside le user group GUIDE-SHARE France-Groupe Informix www.gsefr.org *et fait partie du directoire de l'IIUG.*

Un petit peu d'histoire

En 1985, les langages de programmation appartenaient à plusieurs familles que l'on connait encore aujourd'hui : C, Pascal, Cobol, Basic... Chacun avait ses principes et ses objectifs, mais, à part Cobol, aucun n'était très orienté vers les applications "business". Cependant, Cobol était très orienté fichier et aucun langage ne semblait s'intéresser aux bases de données relationnelles qui commençaient à apparaître.

Informix, fondée sous le nom de RDS (Relational Database Systems) en 1980, se devait de sortir un langage de développement qui permettrait à ses utilisateurs de tirer parti des données. Dans le milieu des années 80, RDS se renomme Informix Software, entre en bourse et sort 2 nouveaux produits : Informix-SQL destiné aux non-développeurs et Informix-4GL destiné aux développeurs.

Informix-4GL est alors un langage compilable qui permet de faire des applications mode caractère sur des terminaux passifs et des rapports imprimables.

Cette stratégie va permettre à Informix de créer un véritable écosystème autour de ses bases de données et de ses outils de développements. Cela sera payant pour la firme de Menlo Park jusqu'au milieu des années 1990.

Comprendre où nous en sommes

Les années 90 sont les années de l'explosion du mode client-serveur : tout le monde veut son poste de travail et de jolies applications Windows qui s'intègrent à Word pour Windows et Excel. Informix rate le virage du client-serveur et de Windows en lançant NewEra.

Pendant très longtemps attendu, NewEra est censé apporter une réponse à la demande des utilisateurs 4GL pour dynamiser et rajeunir les applications mode caractère 4GL. Malheureusement, le langage introduit des concepts objets, une incompatibilité avec le code 4GL existant et, surtout, n'est pas fiable. Cependant, l'acquisition d'Illustra en 1995, un des pionniers des bases de données objet (presque le NoSQL actuel), puis les problèmes de management à l'intérieur de la société, vont conduire Informix à se recentrer sur les bases de données et laisser de côté les outils.

Comme le langage évoluait peu, plusieurs sociétés ont vu le jour dès le début de cette décennie pour pallier à ce manque. Four J's Development Tools (désormais Four Js) et Querix se sont lancées dans une compétition forte pour récupérer la clientèle désorientée par la stratégie d'Informix.

Vers 1999, les deux éditeurs seront rejoint par Aubit4GL une solution open source toujours supportée. D'autres entreprises se développèrent, comme Art-in-Soft voudra convertir le code Informix 4GL en Java (ça n'a jamais marché), ou encore Awoma qui voudra reprendre les paradigmes d'I-4GL pour augmenter la productivité des développeurs Java dans son framework ThinStructure (ça, ça marchait). Le rachat d'Informix par IBM, en 2001, a mis un terme aux ambitions d'un langage de programmation de quatrième génération aux couleurs d'Informix.

Un modèle économique d'un autre temps

Aujourd'hui, le marché est divisé essentiellement entre Four Js et Querix. IBM maintient une version pour ses clients historiques et la version Open Source semble stagner. Ce marché est essentiellement un marché de maintenance, malgré les innovations apportées par les deux leaders. Il faut également comprendre qu'ils sont restés sur un modèle économique où le développeur paie une licence de développement puis des *runtimes* par utilisateur lorsqu'il déploie. Ce modèle a été durement mis à mal par des outils comme Visual Basic qui faisait sauter le coût du *runtime*, puis par Java qui offrait le compilateur. Eclipse a donné le coup de grâce (et a tué Borland, mais c'est une autre histoire). A noter que Querix utilise Eclipse comme IDE pour sa version de 4GL...

Les spécificités d'Informix 4GL

Au premier examen, Informix 4GL ressemble à C par ses aspects structurels, et Pascal par sa syntaxe. C'est surtout un langage facile à comprendre pour qui sait lire l'anglais, rendant sa maintenance aisée et peu coûteuse.

Un véritable langage

Informix 4GL est un véritable langage procédural et structuré, offrant flexibilité et potentiel fonctionnel que n'ont pas les générateurs de formes. On

retrouve les instructions de contrôle de flux habituelles comme WHILE, FOR, IF, CASE, CALL, RETURN.
Le code-source est consigné dans des modules. Un programme peut être composé par un ou plusieurs modules qui seront compilés puis linkés pour devenir un binaire exécutable. Un programme contient une fonction MAIN et peut contenir un nombre illimité de fonctions. I-4GL supporte l'utilisation de librairies, qu'il est possible de développer soit en 4GL, soit en C. Les variables doivent être déclarées. Leur portée peut être globale, locale au module ou locale à la fonction.

Langage SQL intégré

SQL est intégré à la syntaxe d'I-4GL, permettant au code procédural de manipuler tables et colonnes avec des variables. Celles-ci peuvent être mappées à partir des tables/colonnes, grâce à la commande DEFINE variable LIKE table.colonne ou DEFINE structure RECORD LIKE table.*, très pratique lors d'un changement de schéma de table.

Interface utilisateur dans des fichiers distincts

L'interface utilisateur est définie dans des fichiers de définition d'écrans séparés qui seront affichés par les ordres OPEN FORM ou OPEN WINDOW situés dans le module. Les champs peuvent être mappés sur le contenu des tables, avec possibilité d'y ajouter des attributs.
Cette manière de procéder permet une plus grande facilité de maintenance notamment lorsqu'on ajoute ou supprime des champs, limitant les modifications à apporter au code-source.

Interaction formes d'écran/code procédural

Les objets déclarés dans les formes sont manipulés par le code des modules grâce à des instructions spécialisées comme INPUT BY NAME ou INPUT ARRAY VariableTableau FROM TableauEcran, Ces instructions gèrent les entrées sorties de façon évènementielle, avec les clauses BEFORE/AFTER INPUT, BEFORE/AFTER FIELD, BEFORE/AFTER ROW et ON KEY permettant un contrôle très complet de la saisie et de l'affichage des données. A noter la commande MENU gérant les options de menu, et le très puissant CONSTRUCT BY NAME permettant de construire des requêtes SQL dynamiques.

Un générateur d'états plus puissant qu'il n'y paraît

"REPORT" est un type de fonction conçu à l'origine pour gérer les impressions soit en fichier soit vers une imprimante. Au-delà des fonctions habituelles d'un Report Writer, son intégration avec la couche SQL en fait un outil très efficace pour l'écriture de programmes batch performants, tout en rationalisant radicalement le code-source.

Une ouverture sur le monde

Même si IBM a décidé de "ne pas donner suite" au développement d'I-4GL, le langage a été repris intégralement par trois éditeurs qui l'ont rapidement fait évoluer vers un environnement graphique, devenu obligatoire pour survivre.
Misant sur l'investissement en développement des clients, ils ont rendu possible la transformation d'applications opérationnelles et fiables en applications "modernes", tout en gardant le même code-source.
L'architecture 3-tiers utilisant HTML5 permet à ce jour la possibilité d'exécuter ces mêmes applications sur un client léger, sur un navigateur internet ou sur appareil mobile. I-4GL peut désormais d'invoquer des méthodes Java, rendant possible la manipulation d'objets sortant du champ habituel de SQL. Web Services et Micro Services peuvent être développés directement en 4GL. L'intégration d'autres langages populaires devrait voir le jour d'ici peu, de même que l'architecture nécessaire à exécuter ces programmes dans un environnement Cloud.

Un exemple typique

```
FUNCTION saisie_contact(code)
DEFINE code LIKE param.code
DEFINE r_contact RECORD LIKE contact.*
OPEN WINDOW f_contact AT 5,2 WITH FORM "f_contact" ATTRIBUTE(BORDER)
INPUT BY NAME r_contact.*
ON KEY(control-z,F9)
EXIT INPUT
AFTER FIELD datnaiss
    SELECT * INTO r_contact.
    FROM contact
    WHERE nom = r_contact.nom AND prenom = r_contact.prenom
  AND datnaiss = r_contact.datnaiss
  IF status = 0 THEN
error "Ce contact existe déjà, saisir un autre"
next field prenom
 END IF
AFTER INPUT
  WHENEVER ERROR CONTINUE
  INSERT INTO contact VALUES r_contact.*
  WHENEVER ERROR CALL gestion_erreur
  LET statut = status
END INPUT
RETURN statut
END FUNCTION
------------------------------------------------------------------------
fichier forme d'écran f_contact.per
database leres
screen
{
 Prénom [f03                ] Nom [f004              ]
 Date de Naissance [f005        Rue [f006                ]
 CP   [f007   ] Ville [f008                          ]
}
end
tables
contact
attributes
f003  = contact.prenom,required;
f004 = contact.nom,required,upshift;
f005 = contact.datnaiss,required;
f006 = contact.rue;
f007 = contact.cp, required;
f008 = contact.ville,required;
```

Pour aller plus loin :

Pages Wikipedia sur l'histoire des langages de programmation (http://bit.ly/1R4Od01), sur les bases de données relationnelles (http://bit.ly/1MLo1Q4), l'histoire d'Informix Corporation (http://bit.ly/1NWzikn).
Informix 4GL : "Programming Informix SQL/4GL: A Step-By-Step Approach" (http://amzn.to/1OxYtH6),
Informix NewEra : "A guide for Application Developers" (http://bit.ly/1IJ7xgy),
Wiki Informix français: https://goo.gl/tVWtuZ
Communauté Informix internationale : IIUG - International Informix Users Group (http://iiug.org).
Communauté Informix Francophone : https://goo.gl/h9fBmL
Aubit 4GL : http://www.aubit.com/
FourJs : http://4js.com/
Querix : http://www.querix.com

Les questions qu'il ne faut pas poser en entretien avec un codeur

Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € Autres pays : nous consulter.
PDF : 30 € (Monde Entier) souscription 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

Secrétaire de rédaction : Olivier Pavie

Ont collaboré à ce numéro : V. Loquet, S. Saurel, L. Guillois

Experts : F. Hebrard, F. Libaud, J. Bronner, F. Maury, J. Landon, B. Boucard, T. Pierrain, C. Villeneuve, C. Heral, N. Humann, T. Desmas, D. Djordjevic, J. Antoine, G. Burlot, J. Scherrer, M. Bouilloux, J. Giacomini, F. Chevalier, F. Chapuis, J-G Perrin, E. Vercelletoo, CommitStrip

Couverture : © AcidLabs

Maquette : Pierre Sandré

Publicité : PC Presse,
Tél.: 01 74 70 16 30, Fax : 01 41 38 29 75
pub@programmez.com

Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.

Marketing et promotion des ventes :
Agence BOCONSEIL - Analyse Media Etude

Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Contacts
Rédacteur en chef : ftonic@programmez.com
Rédaction : redaction@programmez.com
Webmaster : webmaster@programmez.com
Publicité : pub@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1215 K 78366 - ISSN : 1627-0908

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

L'INFORMATICIEN

DÉVELOPPEZ 10 FOIS PLUS VITE

WINDEV® 21

TDF TECH 2016

VOUS ÊTES INVITÉ !

100% TECHNIQUE

Inscrivez-vous vite !

Montpellier	mardi 8 mars
Toulouse	mardi 15 mars
Bordeaux	mercredi 16 mars
Nantes	jeudi 17 mars
Bruxelles	mardi 22 mars
Lille	mercredi 23 mars
Paris	**jeudi 24 mars**
Strasbourg	mardi 29 mars
Lyon	mercredi 30 mars
Marseille	jeudi 31 mars
Genève	mercredi 5 avril

35 sujets techniques sur WINDEV 21, WEBDEV 21 et WINDEV Mobile 21.

11 villes

du 8 mars au 5 avril

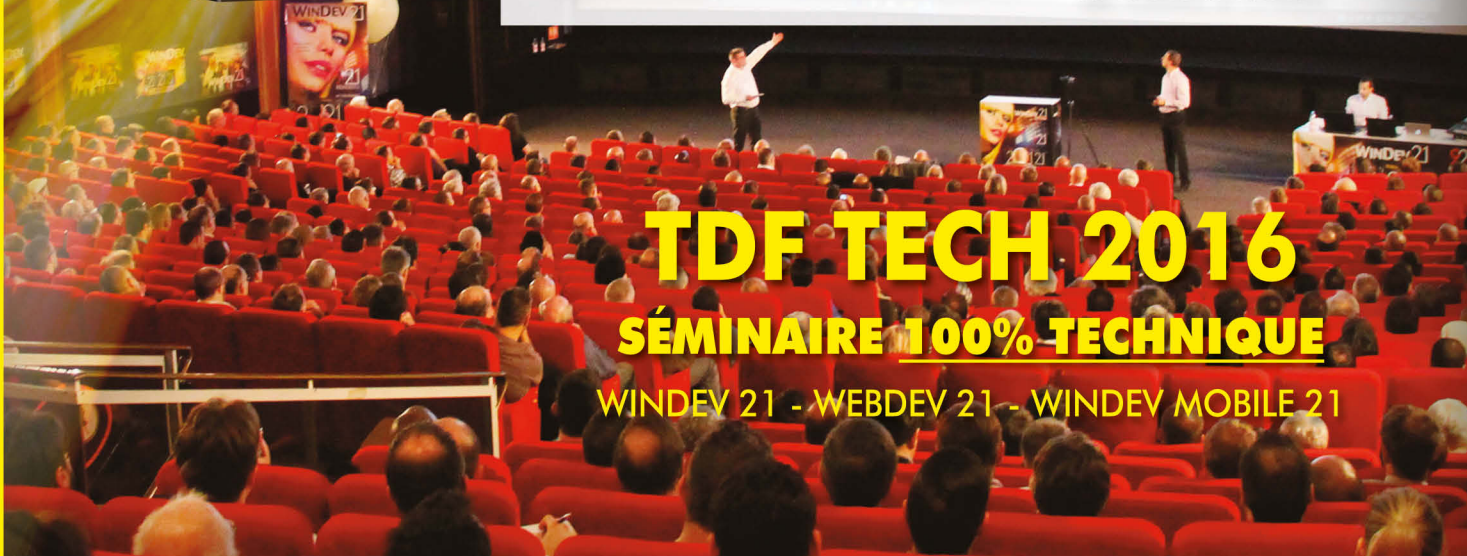
10.000 places

inscrivez-vous vite !

(gratuit)

www.pcsoft.fr

de 13h45 à 17h45



TDF TECH 2016

SÉMINAIRE 100% TECHNIQUE

WINDEV 21 - WEBDEV 21 - WINDEV MOBILE 21