PROGRAMMEZ!

le magazine des développeurs

#204 - février 2017

Le développeur est-il « écologie compatible » ?

Microsoft + Open Source

Pourquoi Microsoft est-il le meilleur ami de l'open source ?

JavaScript

Langage détesté devenu langage star !

Choisir sa plateforme de développement mobile

Quelles compétences pour le développeur 2017 ?

M 04319 - 204 - F: 6,50 € - RD

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

ikoula
HÉBERGEUR CLOUD

PRÉSENTE

CLOUD**IKOULA**ONE

Le succès est votre prochaine destination

MIAMI SINGAPOUR PARIS
AMSTERDAM FRANCFORT _ _ _

*CLOUD**IKOULA**ONE* est une solution de Cloud public, privé et hybride qui vous permet de déployer en 1 clic et en moins de 30 secondes des machines virtuelles à travers le monde sur des infrastructures SSD haute performance.

www.ikoula.com sales@ikoula.com 01 84 01 02 50

ikoula HÉBERGEUR CLOUD

CLOUD | INFOGÉRANCE | SERVEUR DÉDIÉ | VPS | MESSAGERIE

L'obsolescence programmée des développeurs

On parle beaucoup d'obsolescence programmée. Le sujet est sensible et important. Mais nous oublions parfois une obsolescence qui touche directement le développeur : sa propre obsolescence. Nous parlons ici de ses compétences techniques, de son profil technique.

L'obsolescence des compétences n'est pas propre à notre domaine. Elle concerne tout le monde ou presque. Pour un développeur, il est nécessaire de rester en alerte, de regarder ce qu'il se passe, d'aller aux meetups, aux conférences. Un développeur doit se mettre à jour régulièrement, sous peine d'être techniquement en retard. En réalité, il faut trouver un équilibre entre les évolutions et son quotidien. Car il y a un décalage, parfois énorme, entre ce que nous voyons en conférence, chez les éditeurs et le terrain. Oui, la veille technologique ne vous servira pas immédiatement mais cela évitera d'être trop en décalage. Si la zone de confort est toujours appréciable et sécurisante, elle peut aussi vous couper de la réalité de votre métier.

Comme vous le verrez dans notre rubrique carrière du mois, il faut être vigilant sur vos compétences et observer et comprendre le marché et les tendances. Car si vous souhaitez changer de poste, évoluer, mieux vaut avoir une compétence à jour et avoir une idée des tendances sur les langages et sur les frameworks. Si vous souhaitez changer de compétence, mieux vaut ne pas se tromper. C'est pour cela que l'on évoquera aussi les technologies à suivre en 2017 et les possibles orientations.

Nous parlerons aussi dans ce numéro d'un sujet qui nous tient à coeur depuis plusieurs années. Nous avions publié un dossier en mai 2012, n°152, et plus récemment, dans le numéro 178, septembre 2014. Nous revenons en ce début de 2017 sur l'importance du développeur dans un monde technologique plus responsable et écologique. Avec quelques réflexes, quelques bonnes pratiques, il est possible de faire beaucoup.

Bonne lecture

ftonic@programmez.com

Dans le prochain numéro !
Programmez! #205, dès le 3 mars 2017

Souriez ! Vous êtes analysé(e)s
Les API et technologies cognitives se multiplient. Quels usages ? Quelles limites ?

Android Things
Google réinvestit dans les objets connectés.
Adieu Brillo, bonjour Android Things

20 milliards $,

la somme distribuée aux développeurs (Apple).

Acer

réinvente le transportable : 8,8kg, écran 21 pouces… pour seulement 8 999 $.

Pas encore passé au 4K ou au 5K, pas de souci, la **8K** arrivera d'ici 2020.
Avec en prime, HDMI 2.1.

L'usine géante de **Tesla,** pour les batteries, va commencer à produire… +4 milliards $ et une superficie de 900 000 m2, à terme !

Un décompte pour la sortie de **Java 9**?
Fallait oser ! http://www.java9countdown.xyz

Tu n'aimes pas **Tesla,** t'inquiète y'a la toute nouvelle FF91 de Faraday Future. Aucun prix annoncé, un financement incertain, une possibilité de sortie en 2018. Promis, le prochain concurrent de Tesla sera la Watt.

iPhone

a été présenté par Steve Jobs il y a 10 ans. Déjà 10 ans !

Microsoft

se lance dans la voiture avec des services connectés prévus pour cette année. L'éditeur ne cherche pas, pour l'instant, à être concurrent frontal de Google ou d'Apple.

UNE VOITURE OPEN SOURCE CO-CRÉÉE PAR RENAULT

C'est une des surprises du grand salon CES 2017. Renault a dévoilé une plateforme automobile simplifiée, basée sur sa gamme Twizy. Le constructeur collabore avec OS Vehicle, spécialisé dans le logiciel automobile. Mais surtout, il s'agit d'un projet open source qui répond au nom de POM. Renault fournit la plateforme proprement dite et les partenaires actuels, OSVehicle et ARM, fournissent les logiciels embarqués et l'électronique. POM a l'ambition d'être capable de passer à la production industrielle et de proposer tous les éléments nécessaires en open source. Pour le moment, deux modèles seront proposés : POM 45 et POM 80, la différence se fait sur la batterie et la vitesse (45 ou 80 km/h).

Le créateur du langage **Swift** quitte Apple pour rejoindre Tesla…

Alexa,

l'assistant d'Amazon, se répand peu à peu chez Amazon et surtout chez les constructeurs en tous genres

Que faire quand un réfrigérateur sous **Windows** se met à jour et que tu ne peux pas prendre de l'eau ? Tu attends ! Ça donne envie d'avoir des objets connectés dans la cuisine…

Nokia a sorti un nouveau smartphone Android, le Nokia 6.

Marissa Mayer avait de grandes ambitions pour sauver Yahoo!. Début janvier, elle a démissionné du conseil d'administration. Son départ sera effectif quand le rachat de nombreux atifs de Yahoo! par Verizon sera totalement réalisé. Et l'éditeur va aussi changer de nom. La fin d'une époque.

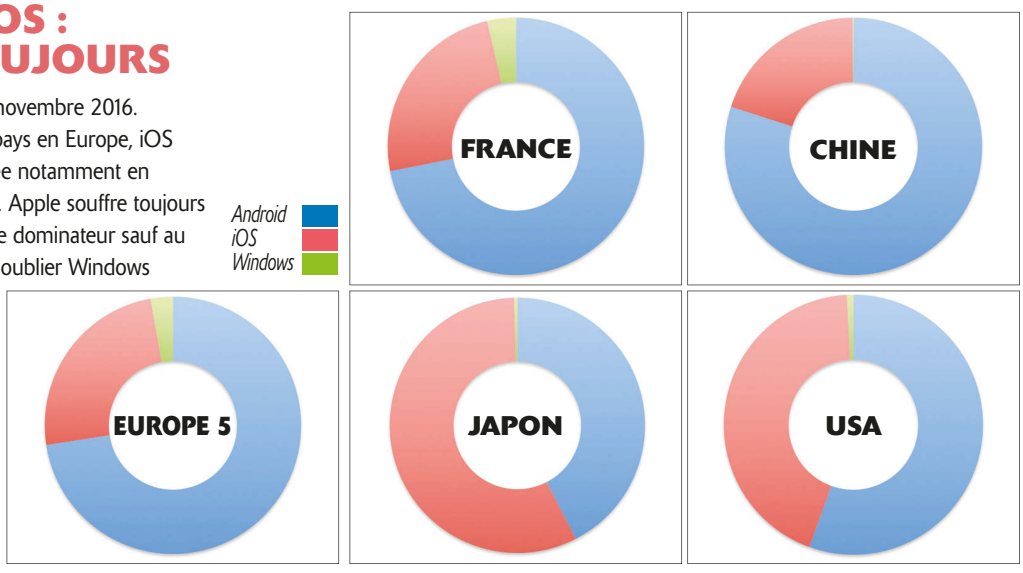
Parrot est en difficulté. Son drone avait beaucoup plu au grand public malgré un prix élevé. Aujourd'hui, la concurrence est très forte sur les drones et le constructeur veut se recentrer sur le professionnel et allège ses effectifs.

IBM

a déposé plus de 8000 brevets en 2016.

ANDROID ET IOS : ENCORE ET TOUJOURS

Kantar a donné les chiffres pour novembre 2016. Globalement, sur les principaux pays en Europe, iOS progresse un peu : belle remontée notamment en France mais baisse en Allemagne. Apple souffre toujours en Chine. Android continue à être dominateur sauf au Japon. Désormais, nous pouvons oublier Windows Mobile qui connait un destin à la Blackberry. Si Microsoft veut sauver son système mobile, il faut soit inonder le marché de téléphones compatibles, soit sortir un vrai terminal capable de s'imposer sur un marché ultra dominant sur le milieu et de haut de gamme.



[2017]

WINDEV 22

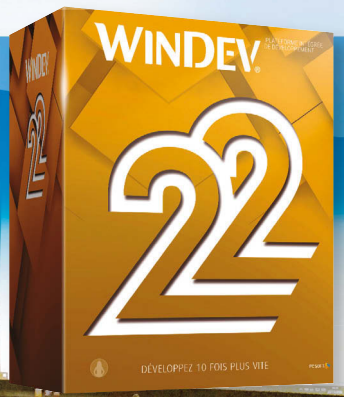
922 NOUVEAUTÉS TECHNOLOGIQUES INDISPENSABLES

La nouvelle version 22 de WINDEV est la version **la plus riche** en nouveautés.

Parmi les 922 nouveautés vous bénéficierez de nouvelles technologies indispensables : ● **Le nouveau champ Traitement de Texte** permet de créer et manipuler des documents **sans sortir de l'application** (et également de les gérer en WLangage) ● Le nouvel **éditeur d'images** orienté développeur permet de retoucher vos images, créer vos icones et vos images «5 états» **sans quitter l'environnement** ● 22 nouveautés boostent les **tables** ● 11 nouveautés boostent vos **plannings** ● **Les nouveaux graphes** ● **IOT** (Objets connectés): support de la norme MQTT ● Pour améliorer la **vitesse des requêtes, HFSQL** trouve les meilleures clés d'index de chaque serveur en exploitation ● **HFSQL** bénéficie d'un **nouveau tableau de bord** ● Installez votre **GDS dans le cloud** en 3 clics pour **2 Euros par mois** ● **Nouveau GDS visuel:** gérez les branches d'un clic ● Les centaines de nouvelles fonctions **WLangage** ● **Le Code Coverage** affiche le pourcentage de code testé ● Le **GO** de projet WINDEV Mobile dans WINDEV ● Dans **WINDEV Mobile 22**, vous êtes averti immédiatement si une ligne de code que vous tapez ne fonctionnera pas sur un système ● Depuis WEBDEV, utilisez des composants **Angular JS**, **Bootstrap**, **jQuery UI**... ● **Télémétrie sur mobile** ● Créez des **Webservices REST** ● Créez des sites complets dans une seule page (**Single Page Application**) ● Utilisez les mots de passe de **Facebook**,... comme identifiants de vos applications et vos sites ● **WebSocket**: c'est le serveur Web qui envoie lui-même les données modifiées aux pages ● Intégrez automatiquement les **trackers Google Analytics** dans vos sites ● Etc, Etc... (liste exhaustive des nouveautés dans la revue de 92 pages accessible sur pcsoft.fr)

Tél Paris: 01 48 01 48 88
Tél Montpellier: 04 67 032 032

WWW.PCSOFT.FR



AFFLUENCE RECORD DANS LES 12 VILLES DU «WINDEV TOUR 22»

Merci de votre fidélité

février

RETROGAMING PLAY 2017

18 & 19 février / Meaux

A 30 minutes de Paris, Meaux va accueillir un des plus gros évènements français de retrogaming, le RetroGaming Play, organisé par l'association RGC, avec Replay et ChezmoA. Ce festival sera l'occasion de mettre en avant la culture du jeu vidéo, avec au menu : des expos, du Freeplay, des animations, de l'arcade, des tournois, des Quizz, un espace vente, des conférences, et, bien sûr, une grande dose de passion ! Parmi les présents et les conférenciers : Bertrand Brocard, AHL, Florent Gorges, Douglas Alves ou bien Abrial Da Costa et bien d'autres ! Il y aura aussi sur place un pôle Youtubeurs/Streamers afin que cette nouvelle façon de vivre le jeu vidéo soit aussi représentée !

Programmez ! est partenaire de l'évènement

Toutes les infos : http://rgplay.fr

mars

BIG DATA PARIS 2017

6 & 7 mars / Paris

Cette 6e édition parlera beaucoup d'Intelligence Artificielle et de Machine Learning. Beaucoup de choses à voir :

- **170 Exposants** français et internationaux viendront dévoiler leur savoir-faire et solutions au travers d'une exposition d'envergure au Palais des Congrès.
- **L'ensemble des fournisseurs du marché français** à retrouver sur l'exposition. Préparez votre visite dès à présent !
- **Plus de 12 500 professionnels** de la donnée et décideurs stratégiques se sont déjà donné rendez-vous pour le congrès : une occasion unique d'échanger avec les leaders et les nouveaux challengers du Big Data et de rencontrer les partenaires de vos projets.
- **Des parcours experts :** anticipez les dernières avancées technologiques de l'écosystème Big Data autour d'ateliers dédiés.
- **Le village Start-up :** venez rencontrer les pépites de la scène tech française sur un espace dédié à l'innovation et aux technologies de rupture.
- **Le lab AI :** partez à la découverte de 15 acteurs les plus pointus autour de l'intelligence artificielle pour un parcours immersif inédit.
- **Le trophée de l'innovation :** les entreprises et startups du monde entier sont invitées à dévoiler aux membres de ce jury d'exception leurs projets Big Data les plus innovants et performants !

Pour en savoir plus : www.bigdataparis.com

GAME OF CODE

11 & 12 mars / Luxembourg

Relevez le défi de programmation du Game of Coding, un grand hackathon qui se déroulera au Luxembourg. C'est une compétition de 24h avec défis, coaching mais aussi l'occasion de voir du monde, d'autres dévs et de s'amuser un peu (ou beaucoup). Programmez ! est partenaire de l'évènement

Toutes les infos : http://www.gameofcode.eu

A venir

JOURNÉE FRANÇAISE DES TESTS LOGICIELS :

11 avril / Montrouge

Le Comité Français des Tests Logiciels organise la 9e édition de la Journée Française des Tests Logiciels, le 11 avril prochain au Beffroi de Montrouge. Plus de 900 professionnels sont attendus sur cet événement, qui s'est au fil du temps imposé comme l'événement de référence du test logiciel en France. 16 conférences, Keynotes, et retours d'expériences rythmeront la journée autour d'un fil conducteur : « Le test logiciel au cœur de la maîtrise des risques de la transformation digitale des entreprises ». Cette édition sera l'occasion pour le CFTL de communiquer les résultats de l'Observatoire 2017 des Pratiques du Test en France. Une journée de tutoriels sera proposée la veille de l'événement, le lundi 10 avril. Les professionnels souhaitant approfondir leurs connaissances des bonnes pratiques du test, seront accueillis au Beffroi de Montrouge par une équipe de formateurs expérimentés, praticiens reconnus dans le domaine des tests logiciels, qui proposeront des enseignements pratiques sur les thématiques essentielles du test. Les inscriptions se font exclusivement en ligne sur http://www.cftl.fr/JFTL/accueil/

CFTL Journée Française des Tests Logiciels 2017
Comité Français des Tests Logiciels
ISTQB

PHP TOUR 2017

18 et 19 mai / Nantes

La grande conférence itinérante PHP s'arrêtera cette année à Nantes !

COMMUNAUTÉS

JUG Nantes :
1er février, soirée RxJava, Spring 5, Ratpack et Couchbase.

JUG Toulouse :
8 février, Functional Web Applications.

ElsaasJUG Strasbourg :
21 février, Java is going nuts/native.

Docker Grenoble :
27 février, meetup sur Docker 1.13.

Python Grenoble :
22 février, Elasticsearch et Django.

CocoaHeads Strasbourg :
9 février, sujet non connu.

GDG Nantes :
23 février, Google HashCode 2017.

MUG .Net Nantes :
15 février, utilisation des MS Cognitive Services avec Xamarin.Forms.

Software Craftsmanship Bordeaux :
13 février, Coding Dojo.

MUG .Net Toulouse :
20 février, .Net Standard et outils pour les dévs.

Docker Lille :
9 février, open space Docker.

BIG DATA by CORP. PARIS 2017

Congrès & Expo

6e édition

Rendez-vous les
6 & 7 mars 2017
Palais des Congrès

+12 000 participants
+200 exposants
+250 intervenants

Participez à l'**événement leader** en France et **préparez l'avenir** Big Data de **votre entreprise**

Master the Data Galaxy

www.bigdataparis.com by CORP. in Corporations we Trust

CES 2017 : des robots et objets pour apprendre à la fois la robotique et à coder

• François Dursus
Creative Developer
Le Lab SQLi

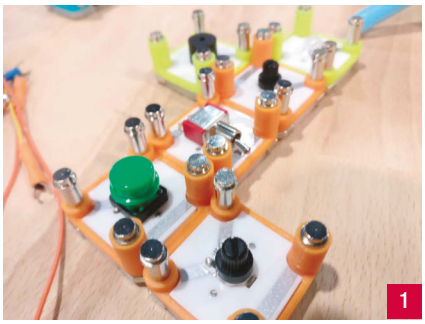
Depuis quelques années nous assistons peu à peu à une démocratisation du "code" auprès du grand public. Le but ? Mettre à mal les a priori sur la programmation informatique comme discipline réservée à une population « geek ». Même si on en est encore loin, on parle de plus en plus de la possibilité d'enseigner les bases de la programmation à l'école, un peu à l'instar de n'importe quelle langue vivante. Nous avions publié début janvier, plusieurs reportages sur les annonces, les produits et projets présentés durant le CES 2017 à Las Vegas. Nous vous proposons quelques autres projets orientés enfants et ados.

Circuit Cubes par Tenka labs : comprendre l'électronique

Ces derniers ne sont plus vraiment en lien direct avec la programmation, du moins dans l'état actuel du projet, mais plus de l'électronique, puisqu'il va s'agir ici de petits blocs assemblables ayant tous une utilité différente.

Un bloc pourra être un simple bouton poussoir, l'autre un potentiomètre, un autre une lumière, un buzzer, un vibreur, un moteur ou encore un capteur de mouvement et enfin bien évidemment un bloc d'alimentation pour faire fonctionner le tout. [1]

Chaque bloc est aimanté rendant leur assemblage extrêmement simple avec la possibilité de les lier par câbles si deux modules doivent être distants. En mettant en série le bloc d'alimentation, le bloc de bouton poussoir puis le bloc de lumière, l'enfant pourra réaliser son premier circuit électrique qui allumera la lumière en appuyant sur le bouton poussoir. En jouant avec l'assemblage il pourra aussi faire tourner un moteur plus ou moins vite via le potentiomètre, allumer la lumière selon l'inclinaison de son montage etc. Voici un exemple très simple de robot se déplaçant par le biais du vibreur connecté au bloc d'alimentation : [2].


1


2

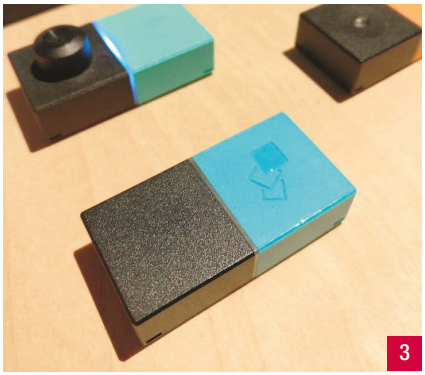
Leur site est, à l'heure actuelle, accessible uniquement par le biais d'un mot de passe *(récupérable par simple demande par mail)*, mais montre plusieurs exemples de constructions ; pour chacun, un tutoriel étape par étape, afin de construire une voiture, une lampe torche, un robot dessinateur, un quadripode façon TB-TT de Star Wars et bien d'autres.

Leur vidéo de présentation montre plus en détails ces objets et laisse présager qu'un module Bluetooth devrait voir le jour bientôt, même s'il n'a pas été montré au CES :
https://www.youtube.com/watch?v=9kXwES-d52o

Mesh par Sony : les « dominos » Bluetooth

Si le projet Mesh n'est pas nécessairement ciblé sur les enfants, il ne semble pour autant pas du tout absurde de les laisser jouer avec. Il se compose de petits blocs en forme de dominos ayant tous une fonctionnalité particulière. Présenté comme ça, on se dit que cela ressemble beaucoup au projet précédent des Circuit-Cubes mais leur usage reste en réalité assez différent.

La différence de base étant que les dominos ne se connecteront pas entre eux physiquement


3

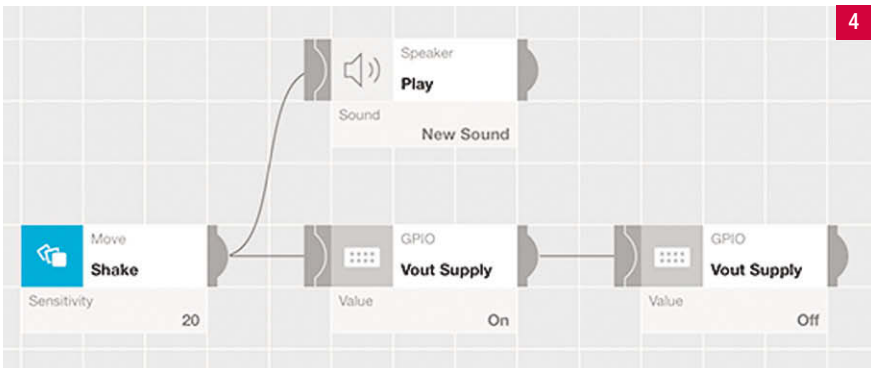
mais via Bluetooth ce qui, de fait, va demander un logiciel permettant de les lier. [3]

Sept modules sont actuellement disponibles et permettront ceci :

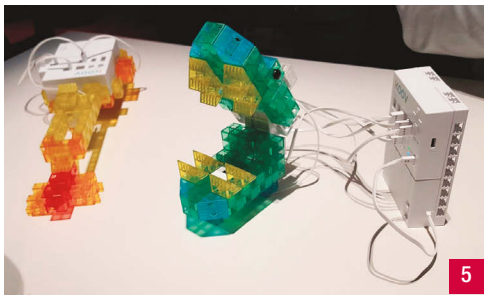
- Allumer une LED RGB ;
- Déclencher une action par pression ;
- Détecter un mouvement humain ;
- Détecter le déplacement d'un objet ;
- Détecter un changement de luminosité ;
- Capter la température et l'humidité ;
- S'interfacer avec des objets électroniques (GPIO).

Le module le plus intéressant étant le dernier puisqu'il permettra, par exemple, de faire tourner un moteur, allumer une lumière, allumer la machine à café, ou autre, en fonction de ce qu'il se passe sur les autres modules.

La plupart de leurs exemples réellement intéressants utilisent en effet ce module ; l'inconvénient étant que son utilisation demandera un



4


5

minimum de connaissances en électronique. Ces modules sont liables entre eux via une application sur tablette qui affichera tous les modules disponibles et proposera de créer des liens. [4]

Un projet plutôt sympathique donc mais avec un inconvénient de taille, son prix. Chaque module va en effet coûter entre 40$ et 60$.

Koov par Sony, moins accessible mais plus de possibilités

Sony s'est aussi lancé dans ce filon en proposant un écosystème très similaire à celui de Lego précédemment cité mais en moins poussé d'après l'aperçu que l'on a eu. L'application est bien moins accessible que celle présentée par Lego et ne propose que la partie programmation là où Lego va accompagner l'utilisateur de la construction à la programmation et à l'ajout de comportements. Voici un exemple montré lors du CES, une gueule de crocodile dans laquelle nous sommes invités à poser notre doigt et le retirer avant qu'il ne claque des dents. [5]

Comme on peut le voir sur cette photo, l'intégration du "cerveau" de l'objet est loin d'être subtile contrairement à Lego Boost. En revanche, il possède bien plus de connecteurs possibles ce qui l'approche plus du Lego Mindstorm. Voici une vidéo de présentation : https://www.youtube.com/watch?v=sNeN-yZHT5E •

Prises et onduleurs : un équipement de base à installer

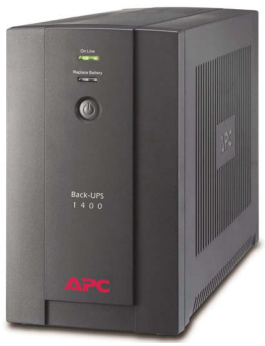
Trop souvent encore, nous oublions les règles simples de protection électrique des matériels que nous utilisons chaque jour : PC, portables, routeurs, imprimantes, écrans, etc. Deux éléments doivent être installés : les prises surtensées et les onduleurs. Chacun joue un rôle spécifique dans votre installation.

La prise surtensée est le minimum et évitera des dégâts potentiellement importants. Comme nous l'a précisé Yves Dutang (directeur IT distribution France, Schneider Electric), il s'agit de protéger les alimentations, et donc le matériel, contre des surtensions. « La sous ou surtension peut avoir des conséquences », précise Yves Dutang. Par exemple, en cas de foudre, une brusque surtension peut se produire sur votre installation électrique et le réseau.

L'onduleur va jouer un autre rôle, en particulier « permettre aux utilisateurs de travailler ou d'éteindre (proprement) le matériel » indique Yves Dutang. Le premier usage de l'onduleur est celui de procurer une autonomie de fonctionnement en cas de panne du réseau électrique. Les onduleurs « généralistes » soutiennent entre 5 et 15 minutes d'autonomie, selon le modèle et les matériels branchés. Ils vont aussi permettre de lisser le courant qu'il soit en sous ou surtension. Et surtout, il va pouvoir gérer à la volée toutes les microcoupures qui peuvent intervenir sur le réseau électrique, souvent quasi imperceptibles pour l'utilisateur alors qu'elles sont réelles. Vous vous en rendrez compte uniquement par l'activité de l'onduleur. Or ces microcoupures peuvent, comme une surtension brusque, endommager le matériel. C'est comme cela que nous avons grillé des disques durs et même des écrans. Sur les constructions récentes, la qualité de l'installation est meilleure, mais tout de même. Nous avons eu une mauvaise expérience avec des CPL qui ont cramé suite à une panne électrique. Et ne dites jamais : cela n'arrive qu'aux autres ! Dans les onduleurs, il existe deux principales technologies : off-line et on-line. En off-line, si la tension varie, les batteries compensent. La mise en action est rapide.

La technologie on-line convertit en permanence le courant, l'onduleur fournit l'électricité uniformément à tous les matériels connectés. Une troisième technologie existe : Line-interactive. Cette approche améliore l'off-line avec des temps de réaction plus courts. On parle de millisecondes.

Attention à la puissance



Il existe plusieurs constructeurs et de nombreux modèles. Trois critères peuvent être pris en compte :

- La technologie : on-line, off-line ;
- Le nombre de prises ;
- La puissance en VA (Volts Ampères).

Il faut que la puissance VA de l'onduleur supporte les matériels branchés depuis. Vous trouverez souvent le calcul suivant : Watts de votre matériel * 0,66. La valeur 0,66 varie jusqu'à 0,9. Par exemple, pour un Dell Alienware 17 R3, la consommation peut monter à 190 Watts en forte charge. Rajoutons un écran 5K LG, soit 200 W en charge maximale. Enfin, rajoutons un NAS de type Synology DS216, soit env. 16 W. Nous aurions théoriquement une consommation en pleine charge d'environ 406 W. Il faudra alors un onduleur avec une puissance d'au moins 270 VA. Mieux vaut surdimensionner la VA pour améliorer la tenue des batteries. Si vous cherchez une autonomie « longue », vous devrez déployer un onduleur à fortes capacités. Et n'oubliez pas de bien le configurer pour optimiser son usage. Et il faut aussi changer les batteries tous les 3-4 ans, en moyenne. Vous ne pourrez plus dire que vous ne saviez pas. •

TRIBY IO

Invoxia va prochainement proposer son objet connecté triby IO qui pourra diffuser de la musique, mais aussi s'intégrer plus largement à la domotique et aux objets de votre environnement notamment en supportant IFTTT et le service Amazon Alexa.

TI INNOVATOR HUB

Texas Instruments annonce pour le monde de l'éducation un nouveau kit : TI Innovator Hub. Il doit aider à la découverte de la programmation et de l'électronique. On peut ainsi rapidement créer des petits projets. On dispose d'une planche à pain, d'un boîtier avec sa carte électronique, de ports I2C. Il est compatible avec les plusieurs calculatrices TI. La programmation se fait en Lua ou en TI Basic.

UN SMARTPHONE EN MODE PC

Miraxess sortira en octobre prochain un ordinateur portable un peu spécial : vous connectez un smartphone et il se « transforme » en PC de travail. Pour le moment, Mirabook sera compatible uniquement Android et Windows Mobile. Il s'appuie sur les fonctions Continuum de Windows et Andromium d'Android. HP avait déjà ambitionné la même approche, sans véritablement convaincre.

CHRONIQUE : LA VR OU MIXTE, QUELLE RÉALITÉ MARCHÉ ?

Révolution, bouleversement des usages, star des fêtes de fin d'année, etc. On a tout entendu ou presque sur les casques de réalité virtuelle, les vrais et les supports smartphone.
Plusieurs incertitudes existent et on pourrait faire un parallèle avec l'impression 3D qui était partout il y a 2 ans et bien moins aujourd'hui. L'offre actuelle, pour le large public, est limitée, principalement Facebook, HTC, Sony. Je ne mets pas Microsoft avec Hololens car il s'agit d'un casque de niche destiné à un marché professionnel et non grand public. Les casques actuels sont-ils réellement grand public ? Malgré les nombreuses démos visibles, les premières salles VR ? Non, ces casques se destinent encore à un public restraint, même si les salles VR peuvent toucher un public large. Aujourd'hui, nous voyons que ces casques se destinent plutôt aux utilisateurs avancés, geeks et hard gamers. Les tarifs sont encore élevés et nous ne sommes finalement qu'à la première génération de casques.
Plusieurs freins demeurent :

- Un temps d'utilisation encore limité : 40 minutes en moyenne, certaines personnes ne dépasseront pas 20 minutes ;
- Prérequis materiel ;
- Les systèmes de positionnement et d'espace pas toujours simples ;
- Un contenu très orienté jeu ;
- L'espace d'action.

Pour le moment, il faut agir aussi bien sur le contenant (le matériel) que sur le contenu. En 2017, l'offre devrait évoluer et l'arrivée des casques compatibles Windows Holographic permettra de faire évoluer le marché. Le tarif attendu se veut attractif par rapport à l'actuel, reste à voir les capacités techniques réelles de ces (futurs) casques. Il faudra aussi définir des spécifications communes, des standards de base, mais pas avant 2018 ou 2019, le marché n'est pas mature. Ces standards concerneront le matériel et le logiciel. Le plus difficile sera de parler le même langage et d'éviter les définitions floues. Sur la partie logicielle, il est important de faciliter la vie du développeur.
Pour nous, le poids est un élément important, ainsi que le design général des casques. Le câble devrait lui aussi trouver des solutions pour le rendre moins encombrant, voire, trouver des solutions pour le supprimer. Il faudra aussi travailler sur la latence et la fluidité même si ces deux points sont déjà très bons.
Si la VR veut s'installer à la maison, il faudra absolument élargir les contenus pour éviter l'effet « on fait quoi avec ? ». Il y a quelques tentatives ici et là, mais elles restent marginales et les éditeurs et fournisseurs de contenus attendent aussi un parc d'utilisateur plus important. L'écosystème est crucial pour le succès ou l'échec d'une plateforme, d'un matériel. Le parallèle avec le smartphone est intéressant car l'explosion des apps sur les App Store a été une des clés du succès. D'autre part, nous constatons aussi l'absence de tout projet open source important, ce qui n'est pas une bonne nouvelle. Le marché naissant et les moyens techniques gênent la création des projets ouverts viables.
Il faudra aussi améliorer l'utilisation sur la durée et mesurer les effets réels sur la santé et les troubles possibles, notamment perte de l'équilibre, vision troublée, etc. Plusieurs questions se posent aussi pour 2017 :

- La durée de vie des casques actuels ;
- Qu'attendre réellement de Magic Leap après beaucoup de promesses, la réalité a été dure pour la start-up qui a avoué avoir « truqué » des démos. Où en est réellement le développement de sa technologie ?
- Apple a communiqué à plusieurs reprises sur la réalité mixte. La Pomme va-t-elle annoncer quelques produits et/ou idées en 2017 ?
- De vrais projets open source et open hardware autour de la VR ?

HOLOJS : UN FRAMEWORK OPEN SOURCE POUR HOLOLENS

Mi-décembre 2016, Microsoft a mis en ligne sur GitHub un framework open source pour Hololens : HoloJS. Il s'agit d'un framework complet pour créer des applications Universal Windows Platform (UWP) basées sur JavaScript et WebGL. Le tout s'appuie sur une librairie C++ de l'OpenGL ES. Le framework s'occupe de mapper l'OpenGL en WebGL dans l'application développée. Le moteur JavaScript, Chakra est utilisé ainsi que le projet Angle. Angle est un projet Microsoft pour mapper les API OpenGL ES en DirectX 11. Il fonctionne sur toutes les plateformes Windows. Durant nos tests, nous avons eu beaucoup de problèmes avec HoloJS provoquant des arrêts systématiques de Visual Studio 2015 et le reboot complet dans la machine. À utiliser avec prudence pour le moment.

Abonnez-vous à programmez!

le magazine des développeurs

Nos classiques

1 an — 11 numéros — **49€***

2 ans — 22 numéros — **79€***

Etudiant — 1 an - 11 numéros — **39€***

** Tarifs France métropolitaine*

Abonnement numérique

PDF — 1 an - 11 numéros — **35€***

Souscription uniquement sur
www.programmez.com

Option :
accès aux archives 10€

Nos offres clés USB

1 an offre 2017 — 59€*

11 numéros
+ 1 clé USB***
+ 4 n° vintage****

** Au lieu de 103,99 € (1 an : 49 €, clé USB : 34,99 €, n° vintage : 20 €) Limitée à France métropolitaine*

Clé USB 4 Go. Photo non contractuelle. Testé sur Linux, OS X, Windows. Les magazines sont au format PDF.

2 ans offre 2017 — 89€**

22 numéros
+ 1 clé USB***
+ 4 n° vintage****

***Au lieu de 133,99 € (2 ans : 79 €, clé USB : 34,99 € n° vintage : 20 €) Limitée à France métropolitaine*

**** Clé USB Programmez! 4 Go contenant tous les numéros depuis le n°100*
***** Selon les stocks disponibles. Antérieur au N°168*

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

☐ *Abonnement 1 an au magazine :* 49 €
☐ *Abonnement 2 ans au magazine :* 79 €
☐ *Abonnement étudiant 1 an au magazine :* 39 €
Photocopie de la carte d'étudiant à joindre

☐ *Abonnement 1 an au magazine :* 59 €
11 numéros
+ 1 clé USB
+ 4 n° vintage

☐ *Abonnement 2 ans au magazine :* 89 €
22 numéros
+ 1 clé USB
+ 4 n° vintage

PROG 204
Offre limitée, valable jusqu'au 2 mars 2017

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : @

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

☐ Je souhaite régler à réception de facture

** Tarifs France métropolitaine*

CES 2017 : la part belle à la réalité virtuelle et à la réalité augmentée

• Edouard Bataille
Creative Developer
Le Lab SQLI
Notre envoyé spécial au CES

En arpentant les allées du CES, il est surprenant de voir le nombre de fabricants de casques de réalité virtuelle, et pourtant, parmi les acteurs principaux, Oculus et HTC n'étaient pas présents directement sur le salon. Seul le Playstation VR était accessible en test sur le stand de Sony.

LES CASQUES DE REALITE VIRTUELLE AU BANC D'ESSAI

Au programme des tests VR sur le stand Sony Gran Turismo en tête de proue aux côtés d'autres démos de jeux. [1]

Sur le nombre incalculable de fabricants de casques présents au CES, peu ont retenu notre attention, mais ça a été le cas pour Panasonic et son prototype qui promet un casque avec un angle plus élargi que les casques traditionnels.

La démo n'était cependant pas aussi impressionnante qu'on aurait pu l'attendre, les démarcations visibles entre le devant et les côtés de l'écran étant bien visibles. [2]

Dans un autre registre Lenovo présente le premier casque à intégrer Windows Holographic, la plateforme de réalité virtuelle et augmentée de Microsoft. Doté de 2 écrans Oled de 1440x1440 pixels, ce casque plus léger que ses concurrents (Oculus et Vive) et dont le design rappelle un peu celui de Playstation VR, est annoncé à moins de 400$. [3]

Coté AR nous avons pu voir le casque de Occipital qui embarque une technologie de reconnaissance spatiale permettant ainsi de faire de la mixed reality (réalité virtuelle et augmentée). [4]


1


2


3

L'Occipital possédait déjà la technologie structure qui comme le Tango de Google permettait de scanner des objets ou de permettre à une tablette d'être conscient de son environnement. Ils ont maintenant fabriqué un casque équivalent à un Google Cardboard mais doté d'une caméra grand angle. Le tout combiné donne un casque de plutôt bonne facture avec des capacités assez étonnantes. Dans la même lignée, nous parlerons du téléphone Asus ZenFone AR. Doté de 8Go de RAM, Tango et Google Daydream ready ce téléphone pousse le cran encore plus loin puisque la technologie Tango est bien plus aboutie que celle d'Occipital. De plus, le support de la plateforme Daydream assure au téléphone un écosystème de réalité virtuelle beaucoup plus complet.

LES LUNETTES DE REALITE AUGMENTEE : LES LENTILLES LUMUS SE DEMARQUENT

Nous avons également pu voir beaucoup de lunette de réalité augmentée avec plus ou moins de succès. Dans tous les prototypes que nous avons vus, les lunettes étaient vraiment sombres et le champ de vision plus ou moins restreint. Nous avons été cependant bluffés par le fabricant de lentilles "Lumus".

Nous avons pu tester ces lentilles sur leur stand et autant dire que nous avons été bluffés à la fois par la qualité de l'image mais aussi par la transparence des verres. [5] Lumus ne fabrique pas les lunettes et nous attendons donc de voir si cette technologie va être intégrée par des gros fabricants et comment.

DES ACCESSOIRES VR : DU SIEGE AU SIMULATEUR DE VOL

Coté accessoires, c'est par contre un peu la déception. Très peu nombreux, les accessoires sont soit trop volumineux soit trop chers pour s'adresser au grand public, à l'image d'un siège qui bouge en fonction de l'expérience VR que vous vivez. Pouvant s'adresser par contre plus au grand public, des chaussures et gants avec retour de force Haptic vous permettant de ressentir à vos pieds et mains les sensations que vous expérimentez en VR. Doté d'un SDK Unity, c'est sans doute l'accessoire qui a retenu le plus notre attention.

UN PC PORTABLE VR VERSION MSI

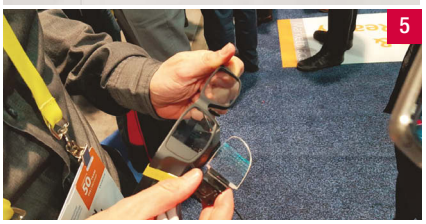
Coté PC, outre les PC portables VR ready, MSI nous a présenté sa bonne idée, le VR One backpack PC, doté d'un processeur Intel i7-6820HK et d'une Nvidia GTX 1060 ou 1070 qui permet d'emporter votre PC sur le dos pour jouer avec votre casque de réalité virtuelle.

Avec 90 minutes d'autonomie, ce pc est muni de deux batteries permettant de changer une batterie à la fois, ofrrant ainsi de prolonger l'expérience encore plus longtemps. Ce PC est déjà disponible en pré-commande et sera disponible fin novembre. Reste à savoir si cette idée est la bonne, en effet coté displaylink et chez HTC, on propose des solutions sans fil avec une latence proche de la milliseconde.

CONCLUSION : UN CES RICHE EN VR

Malgré quelques déceptions (sur la VR), ce CES 2017 a été riche en réalité virtuelle, et l'on voit bien que l'industrie s'empare du marché ; reste à savoir maintenant comment ce dernier va se rationnaliser, sûrement autour des grands acteurs HTC Vive, Oculus, Playstation VR et Google avec Daydream. •

4


5

Développement = écologie-compatible

On parle beaucoup écologie, écoresponsabilité, économie responsable / durable, etc. Dans le monde informatique, nous avons facilement parlé de recylable électronique, optimisation de l'impression, des matériels plus efficaces énergiquement, des serveurs plus performants, etc. Le Cloud Computing est parfois présenté comme un avantage écologique. Ce constat est à pondérer car les datacenters sont de plus en plus volumineux même si de nombreux fournisseurs utilisent des énergies renouvelables. Les nouvelles technologies comme l'impression 3D font gagner ou perdre de l'écoresponsabilité. Les smartphones qui se comparent désormais avec des PC. Il y a aussi la multiplication des objets connectés, des terminaux mobiles qui pèsent sur les infrastructures télécom et bien entendu exigent de l'énergie. La moindre panne est vécue comme un enfer : plus de matériel, plus de connexion.
L'obégiciel touche le PC, le mobile et les sites Web. Aujourd'hui, la moindre mise à jour d'une app mobile dépasse souvent 60-80 Mo, les mises à jour système, bien plus. Et que dire d'un Visual Studio complet qui peut demander jusqu'à 27 Go d'installation ! Le stockage étant de plus en plus important, il favorise cette obésité logicielle et pour le moment, la tendance n'est pas à la décrue.
Et le développeur ? Car il ne faut surtout pas oublier un chaînon important : le développeur. Car les apps consomment des ressources, et de plus en plus. Le développeur joue un rôle non négligeable en optimisant le code, les divers assets, en supprimant les codes morts, etc. Nous faisons ici le point sur quelques bonnes pratiques. La rédaction.

MINI-SONDAGE
PROGRAMMEZ!

Oui, j'optimise mon code, l'interface Web, les images, le poids de chaque page. Je fais attention à tout, la charge CPU, GPU... :	**16%**
Partiellement en optimisant les images ou le code mais je ne maîtrise pas tout	**29%**
Non, faute de temps :	**13%**
Je ne sais pas comment faire. Où sont les bonnes pratiques ?	**42%**

102 votants

À quand le code green ?

Ly-jia Goldstein
Software Craftswoman .Net
Arolla

Nicolas Fabre
Software Craftsman .Net
Arolla

Aujourd'hui on entend parler de Green IT ou d'écoconception logicielle, ce sont des approches principalement focalisées sur l'amélioration de la qualité logicielle et l'optimisation de la consommation des ressources (en économisant du temps de traitement machine par exemple).

Il s'agit d'une approche raisonnable puisque le temps de traitement influe sur la consommation d'électricité, le gain est alors direct et l'impact concret, mais cette approche ne prend en compte qu'une partie du secteur et non l'ensemble du cycle de développement. Les développeurs agissent selon leurs convictions et à leur niveau avec une marge de manœuvres réduite et sans pouvoir influer sur la conception des matériels (processeurs, alimentations...) ou certains processus.

On constate toutefois que certains intérêts convergent, les performances impactant directement le coût d'un projet (nombre de machines nécessaires pour héberger une application Web, expérience utilisateur dégradée), un effort certain existe déjà pour optimiser les applications et ainsi réduire directement leur coût et indirectement l'énergie utilisée. Ceci est particulièrement vrai sur les supports mobiles, où la quantité d'énergie est limitée : en augmentant leurs performances, les terminaux sont moins sollicités et auraient donc une durée de vie plus longue, réduisant alors également l'impact en matières premières pour la fabrication d'appareils de remplacement.

De fait, le développeur est "écologie-compatible" puisque l'optimisation du code a un impact direct sur la quantité d'énergie requise pour faire fonctionner un logiciel. Mais est-ce là une piste suffisamment novatrice et pertinente ?

La plupart des "bonnes pratiques" recommandées pour l'optimisation du logiciel sont d'ores et déjà de bonnes pratiques de développement, admises et bien souvent déjà appliquées dans la mesure où les entreprises y voient un gain direct en réduisant les coûts en équipement et leur facture énergétique...

La formalisation de ces bonnes pratiques au travers notamment du Software Craftsmanship a permis de pousser diverses réflexions pour aller plus loin que l'écriture de code de qualité, et analyser les impacts sur notre façon de travailler.

Il est évident, et pourtant souvent oublié, qu'écrire une ligne de code, même si elle ne va pas en production, est coûteux. En effet cette ligne sera rédigée sur un ordinateur consommant de l'énergie ; pour pouvoir l'écrire il y aura eu des échanges par mails sur le besoin, l'utilisation d'un moteur de recherche pour affiner la problématique et potentiellement une relecture du code déjà existant s'il s'agit d'une évolution... et ce, jusqu'au fonctionnement de l'usine logicielle de la compilation au déploiement.

La pratique agile permet d'adresser plusieurs problématiques.
Une étude du Standish Group [1] estime que seules 20% des fonctionnalités d'une application seraient réellement utilisées. Les méthodologies agiles avec une vision produit, Scrum par exemple, sont efficaces pour éviter le développement de fonctionnalités inutiles.
Elles recommandent notamment une priorisation des fonctionnalités et des cycles de développements courts (appelés itérations) pour livrer au plus tôt. En livrant rapidement, les décideurs sont plus enclins à laisser de côté les demandes à moindre valeur ajoutée. Et cela permet surtout d'obtenir des retours permettant éventuellement de rectifier la roadmap du produit afin de le rendre conforme aux attentes.
Une fonctionnalité développée peut ne pas être utilisée car la spécification n'a pas été interprétée de la même manière par le rédacteur ou le développeur. Une collaboration pendant les itérations entre développeurs, experts qualité et métier va permettre de lever les incompréhensions. C'est ce que préconise le Behavior Driven Development. Cette pratique conseille aussi de mener les discussions autour d'exemples concrets (généralement avec une syntaxe Gherkins) pour lever toutes les ambiguïtés. Exemple avec les happy hours :
- Étant donné une pinte de bière au prix normal
- Quand j'en prends une en happy hour
- Alors je ne paye que la moitié de son prix

Cet exemple est trop vague, et nous allons préférer cette version plus concrète :
- Étant donné une pinte de bière à 4€
- Quand j'en prends une en happy hour
- Alors je ne paye que 2€

En communiquant régulièrement ensemble, un langage commun peut alors émerger, ce qui réduit encore plus les malentendus, c'est l'*ubiquitous language* décrit par le Domain Driven Design.
Un autre levier pour réduire son empreinte énergétique est de réduire le nombre de bugs en production, et donc d'augmenter la qualité du code produit. En effet, la correction de ces anomalies implique des cycles de développements, de tests et de déploiements supplémentaires qui peuvent être évités. Cela se traduit par exemple par un investissement sur les bonnes pratiques de développement, comme TDD.
Pratiquer TDD va permettre d'agir sur la quantité et la qualité du code rédigé : si l'on suit les trois lois du TDD énoncées par Oncle Bob, il est impossible d'écrire du code superflu, et donc possible d'économiser des ressources.

- *Vous ne devez pas écrire de code en production à moins qu'il ne fasse passer un test qui échoue ;*
- *Vous ne devez pas écrire plus d'un test suffisant pour échouer, ou qui échouera à la compilation ;*
- *Vous ne devez pas écrire plus de code que nécessaire pour faire passer le test en cours ;*

- (*The Three Laws of TDD* - Uncle Bob [2])

L'idée est ici clairement de ne pas écrire plus de code que nécessaire. A chaque étape, le test devenant plus spécifique, le code se transforme pour prendre en compte les nouvelles contraintes, chaque transformation étant le minimum nécessaire pour faire passer le test qui vient d'être modifié ; l'efficience de la phase de développement devient alors un levier à part entière de l'optimisation du logiciel.
Uncle Bob formalisera ce cycle de transformations dans son article *The Transformation Priority Premise* [3], celui-ci introduit une série de transformations élémentaires (par exemple *return null* devient *return constant*) décrivant l'évolution du code et sa complexification au fur et à mesure du développement.
TDD est également très puissant par ses phases de refactorisation qui améliorent la qualité du code. Ce sont aussi des moments propices à l'optimisation des performances du code lui-même. La maîtrise des ressources repose notamment sur la connaissance du langage de programmation utilisé, des patterns, et du bon sens. Voici quelques exemples en C#, mais applicables à la plupart des langages :

1. Identifier les bons usages du Eager/Lazy Loading

Le chargement des données en mémoire est une question primordiale pour la gestion des ressources où l'on distingue deux cas :

- Les données sont rarement utilisées et il est préférable d'initialiser les objets le plus tard possible : c'est ce que l'on appelle le Lazy Loading ;
- Au contraire, si les objets sont accédés souvent, le Lazy Loading va être coûteux car il nécessite une vérification supplémentaire à chaque accès pour savoir s'il y a nécessité d'initialiser ou non l'objet. Dans ce cas, nous utiliserons de l'Eager Loading, c'est-à-dire que les objets sont initialisés à la création.

Le pattern du Singleton est appliqué dans les problématiques de multithreading, il peut être implémenté différemment selon le type de chargement souhaité.

En Eager Loading :

```
public class Singleton
{
  private static Singleton instance = new Singleton();

  private Singleton() { }

  public static Singleton GetInstance()
      {
    return instance;
  }
}
```

En Lazy Loading :

```
public class Singleton
{
  private static object myLock = new object();
  private static Singleton instance = null;

  private Singleton() { }

  public static Singleton GetInstance()
     {
    if (instance == null)
          {
      lock (myLock)
                {
        if (instance == null)
                      {
          instance = new Singleton();
        }
```

```
            }
        }
        return Singleton;
    }
}
```

Ici, deux vérifications sont nécessaires car l'objet a pu être créé au moment de poser un lock. Le choix du type d'initialisation n'est donc pas neutre. On notera que cette double vérification est prise en compte dans le langage à partir du framework .NET 4.0 avec la classe Lazy<T> :

```
public class Singleton
{
  private static readonly Lazy<Singleton> instance = new Lazy<Singleton>();

   private Singleton() { }

   public static MySingleton Instance
   {
      get
      {
         return instance.Value;
      }
   }
}
```

2. Utiliser les bonnes collections selon les besoins et leurs complexités respectives

Les langages de programmation proposent tous de nombreux types de collections.

En .NET, c'est la classe List<T> qui est la plus utilisée, et la complexité pour retrouver une valeur sans passer par un index est de l'ordre de O(n), n étant le nombre d'éléments dans la collection.

Si le besoin est de rechercher une valeur dans une grande collection de données alors un Hashset<T> est plus adapté (complexité de l'ordre de O(1)), mais la duplication de données et le tri ne sont pas possibles.

En revanche, si la collection doit subir de nombreuses insertions et suppressions en milieu de liste, la LinkedList (implémentation d'une liste doublement chaînée) sera la plus appropriée pour ne pas être pénalisée par les temps de traitement.

Dans la famille des collections triées, SortedList<TKey, TValue> et SortedDictionnary<TKey, TValue> sont très similaires : ce sont des collections à utiliser quand il est nécessaire d'avoir des données triées, et la complexité de recherche est de l'ordre de O(log n). Elles se différencient par la quantité de mémoire utilisée et la vitesse d'insertion et de suppression d'éléments :

- SortedList<TKey, TValue> nécessite moins de mémoire que SortedDictionnary<TKey, TValue> ;
- SortedList<TKey, TValue> est plus lent que SortedDictionnary<TKey, TValue> sur l'insertion et la suppression d'éléments ;
- A partir d'une liste d'éléments préalablement ordonnée, SortedList<TKey, TValue> est remplie plus rapidement que SortedDictionnary<TKey, TValue> .

3. Ajuster le contexte de traitement des données

Lors de l'exécution d'un logiciel, celui-ci devra à un moment manipuler des données. On peut ainsi identifier, entre autres, quatre contextes pour le traitement des données que nous rencontrons au quotidien dans nos applications :

- Traitement en base de données, très classiquement pour récupérer des données relationnelles, par exemple en appelant une procédure stockée ;
- Traitement en fichier, pour sauvegarder ou traiter des données par batch, c'est typiquement le cas des logs applicatifs ;
- Traitement en mémoire, c'est le cas des variables déclarées dans le code, mais aussi lors de l'utilisation d'un ORM (object-relational mapping) comme Entity Framework ;
- Traitement délégué à un tiers, c'est le cas pour les applications mobiles ayant souvent recours à une API REST pour des traitements côté serveur ou dans une architecture microservice.

Chacun de ces cas est accompagné d'avantages (rapidité de traitement, persistance des données ou répartition de la charge de travail), mais aussi d'inconvénients :

Lorsqu'il y a des échanges sur un réseau, nous sommes amenés à rencontrer de la latence ou des coupures, mais aussi une limitation de bande passante. Plusieurs solutions sont possibles pour agir sur ces contraintes :

- L'utilisation d'un cache au niveau de la couche d'accès aux données ou de la couche métier pour les données fréquemment utilisées mais peu sujettes aux changements permet de limiter les appels à la base de données ;
- Le SELECT * est à bannir, la limitation du nombre de colonnes renvoyées permet de réduire l'impact de la bande passante du réseau en diminuant la taille des données qui y circulent ;
- L'architecture applicative peut être adaptée en privilégiant une base de données documentaire comme MongoDB ou Couchbase lorsqu'il s'agit de données de type clé-valeur principalement en lecture ;
- Lorsque Entity Framework est utilisé, il convient de faire attention au fait qu'une requête Linq n'est évaluée que lorsqu'elle est itérée (par un foreach par exemple) ou avec certaines fonctions retournant une valeur (Count, First ou Max par exemple). Cette différence entre IEnumerable et IQueryable est très souvent la cause d'erreurs de conception qui provoquent l'exécution de requêtes en base de données superflues et donc inutilement coûteuse en ressources.

Lorsque nous utilisons les disques physiques, nous rencontrons de la latence pour l'accès aux disques et nous devons aussi faire attention à l'espace disque disponible, la corruption de données ou l'usure du médium. Nous avons à notre disposition un certain nombre d'outils :

- L'utilisation de Nlog ou Logstash pour gérer les données générées de manière asynchrone, par batch ou même en base de données ;
- Mettre en place un bus de données (Rabbit MQ, Kafka) qui conjointement à des "consommateurs" de données permet de ne plus stocker les données brutes mais uniquement les données transformées ;
- Dans les cas où l'utilisation de fichiers est indispensable, il peut être utile de mettre en place un système réparti de stockage et manipulation des fichiers tel que Hadoop.

Enfin pour le traitement en mémoire, nous devons veiller à ne pas saturer la mémoire inutilement, nous sommes pour cela aidés par le Garbage Collector du Framework qui s'occupera de réallouer la mémoire libérée. Cela passe par quelques bonnes résolutions :

- Etre vigilant à l'utilisation des "using" pour garantir la bonne utilisation des objets implémentant l'interface IDisposable et faciliter le travail du garbage collector ;
- Lors de la mise en mémoire ou en cache de données issues de la base de données, il convient de s'assurer que seules les données nécessaires

y sont envoyées, en étant prudent sur l'utilisation du Include() avec Entity Framework par exemple.

Ces problématiques ont un impact direct sur les performances des logiciels et l'usure des terminaux où ils s'exécutent. Ajuster le contexte de traitement ou de stockage des données est donc primordial pour réduire les connexions réseau, les diverses latences ou les accès aux disques.

D'ailleurs, il est à noter que l'explosion des ventes de disques SSD n'est pas sans conséquence. Pour rappel, les disques SSD sont sensibles au nombre de cycles d'écritures, des cycles intensifs réduiraient alors leur durée de vie et potentiellement celles des terminaux (un smartphone par exemple). Spotify a semé la panique en novembre dernier avec un bug, qui provoquait l'écriture de jusqu'à 10 go de données en une heure risquant de provoquer une usure prématurée des disques.

Les utilisateurs de Windows avaient déjà fait part de leur indignation lors de la campagne de migration vers Windows 10, ils s'étaient alors aperçus que la nouvelle version de l'OS de Microsoft s'était téléchargée sans en informer les usagers "pour une meilleure expérience utilisateur".

De manière générale, il est recommandé de surveiller les temps d'exécution de son application ainsi que les interactions entre les différentes briques applicatives, et si possible d'organiser des tests de charge pour identifier les limites de son logiciel. Pour les plus pointilleux, des outils tels que ANTS Performance Profiler permettent d'avoir des données concrètes, de nombreux IDE proposent également des outils de profiling portant sur l'utilisation du CPU et de la mémoire. Visual Studio propose même un profiler dédié à la consommation d'électricité pour les applications mobiles [4].

Néanmoins, force est de constater, qu'il est indispensable de se demander comment anticiper l'avenir. Nous avons vu que le lien entre code optimisé et contexte écologique était historiquement lié à la gestion d'une infrastructure ou de contraintes fortes des terminaux exécutant les applications, mais il existe aujourd'hui un changement profond dans le développement logiciel qu'il ne faut pas négliger : le Cloud.

L'arrivée du Cloud a renversé notre façon d'aborder la gestion d'une infrastructure. Auparavant la mise à jour de nos parcs de machines (verticalement en améliorant la configuration d'une machine ou horizontalement en augmentant le nombre de machines) nécessitait une planification sérieuse, la prévision des coûts et surtout la certitude que cet investissement ne serait pas à perte. En dématérialisant l'infrastructure, le Cloud offre une flexibilité jamais vue jusqu'ici : la scalabilité en fonction des besoins ponctuels pour avoir la juste qualité de service. Ainsi la mise à jour verticale ou horizontale peut se faire en quelques clics et est réversible, la mutualisation des ressources évite aussi le suréquipement et réduit les coûts par rapport à un datacenter classique [5].

Mais ces avantages ne sont pas sans conséquences, la simplicité et la flexibilité masquent les impacts en termes d'équipement ou de facture énergétique, et masquent donc indirectement les coûts environnementaux. Il n'est pas impossible que les décisions de basculer sur le Cloud ainsi que le choix du partenaire (les principaux acteurs du marché étant Amazon et Microsoft) se fassent sur des critères de coût. Or, en masquant une implémentation derrière un interface, on rend la décision plus complexe et le besoin d'expertise plus important encore, sans quoi il y a un risque de désensibiliser ou déresponsabiliser les entreprises à l'impact écologique d'un logiciel ou d'un choix technique, en balayant le problème par un "il suffit d'augmenter le nombre de VM".

C'est d'ailleurs une des principales préoccupations de Greenpeace qui, depuis plusieurs années, anime une campagne pour faire prendre conscience au grand public du coût de l'informatique dématérialisée ainsi que faire pression sur les grands groupes pour aller vers une transition énergétique durable [6].

Dans sa dernière version du rapport Clicking Green [7], Greenpeace sensibilise sur les choix énergétiques des géants du net : si Facebook et Google ont entamé leur transition et se fournissent à plus de 50% en électricité issue des énergies renouvelables, ce n'est pas le cas pour Microsoft et Amazon Web Services (respectivement 32% et 17%). Le choix du fournisseur de machines virtuelles n'est donc pas anodin. Netflix, en se reposant sur les services d'Amazon et représentant une bonne partie du trafic de téléchargement, contribue ainsi à l'accroissement de la consommation d'énergies polluantes.

Greenpeace souligne également que les pays d'implémentation des datacenters peuvent avoir un impact sur le caractère renouvelable de l'énergie utilisée, comme par exemple en Asie du Sud-Est où l'électricité issue du charbon est majoritaire, Microsoft a d'ailleurs initié des recherches sur l'installation de datacenter sous-marins pour réduire l'impact énergétique lié au refroidissement des serveurs [8].

Déléguer la gestion de son infrastructure ne doit donc pas signifier déléguer sa responsabilité sur la question de la consommation énergétique.

EN CONCLUSION

Le rôle du développeur "écologie-compatible" est alors d'autant plus important : il connait l'impact des technologies et choix techniques mais sait qu'il est avant tout consommateur de ressources.

Il est alors nécessaire de prendre conscience que l'écologie dans le logiciel n'est pas une contrainte supplémentaire mais une formalisation de pratiques très souvent déjà mises en place, en cherchant simplement à les replacer dans le cadre concret des transitions que l'on observe. Y penser précocement dans la phase de réflexion du logiciel, et non comme une action d'optimisation a posteriori de la réalisation, permet d'optimiser l'impact non seulement du logiciel fini mais aussi de l'ensemble de son cycle de vie à travers la conception, la réalisation et la maintenance. •

Bibliographie :

(1) The Standish Group International, Inc. 2014 - http://www.standishgroup.com/sample_research_files/Exceeding%20Value_Layout.pdf

(2) Robert Cecil "Uncle Bob" Martin, 2005 - *The Three Laws of TDD* http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd

(3) Robert Cecil "Uncle Bob" Martin, 2013 - *The Transformation Priority Premise* https://8thlight.com/blog/uncle-bob/2013/05/27/TheTransformationPriorityPremise.html

(4) Microsoft, 2016 - *Analyze energy use in Store apps* https://docs.microsoft.com/fr-fr/visualstudio/profiling/analyze-energy-use-in-store-apps

(5) Greg Deckler, 2016 - *Cloud vs. On-Premises – Hard Dollar Costs* https://www.linkedin.com/pulse/cloud-vs-on-premises-hard-dollar-costs-greg-deckler

(6) Greenpeace, 2017 - *Impact environnemental du numérique : il est temps de renouveler Internet* http://energie-climat.greenpeace.fr/il-est-temps-de-renouveler-internet

(7) Greenpeace, 2017 - *Clicking Clean* http://www.greenpeace.org/international/en/publications/Campaign-reports/Climate-Reports/clicking-clean-2017/

(8) Microsoft - *Microsoft research project puts cloud in ocean for the first time* https://news.microsoft.com/features/microsoft-research-project-puts-cloud-in-ocean-for-the-first-time/#sm.000a6gx7z14zhdf3yzs2f9sqloomn#HxTiAbOotoPYGQLe.97

Concernant les performances avec Entity Framework:

Ben Emmett, 2015 - *Entity Framework Performance and What You Can Do About It* https://www.simple-talk.com/dotnet/net-tools/entity-framework-performance-and-what-you-can-do-about-it/

Le développeur est-il « écologie-compatible »

Olivier Philippot
Directeur Technique Greenspector et expert écoconception des logiciels
@simplygreenit

Le numérique, au sens large, est souvent perçu comme "virtuel" et peu polluant, car il repose sur une impression de "dématérialisation". Pourtant il ne fonctionne que grâce à des matériels qu'il faut fabriquer, et faire fonctionner, et à une énergie électrique qu'il faut produire. Le rapport Smart 2020 a ainsi estimé que les Technologies de l'Information et de la Communication (TIC) sont responsables de 2 % des émissions globales de CO2 sur la planète, soit le même impact que le transport aérien. Selon ce même rapport, ces émissions augmentent en permanence et pourraient doubler, en raison notamment de la croissance vertigineuse des datacenters (notamment avec les services Cloud) et des matériels mobiles (notamment les smartphones qui sont dans certains pays le point d'accès à Internet et aux apps). Or même si les matériels récents sont généralement moins consommateurs, les économies d'énergie qui en résultent sont annulées (voire dépassées) par la multiplication du nombre d'appareils mais aussi la multiplication des apps et des services.

Dans ce contexte, quelle est la place le développeur - si rôle il a - pour réduire cet impact ? Est-ce illusoire par rapport aux multiples contraintes des développements ? Faut-il attendre que le matériel continue de s'améliorer et que la technologie nous aide ? Le logiciel peut-il aider à rendre nos environnements plus « écologie-compatibles » ?

Ecoconception logicielle vs obégicielle ?

L'écoconception est définie comme la "**conception d'un produit, d'un bien ou d'un service, qui prend en compte, afin de les réduire, ses effets négatifs sur l'environnement au long de son cycle de vie, en s'efforçant de préserver ses qualités ou ses performances**" (**Grand Dictionnaire Terminologique**).

Il s'agit donc d'appliquer tout au long du cycle de développement du logiciel, donc l'application proprement dite, une démarche de frugalité (répondre à une demande nécessaire, éviter le superflu), de simplicité (optimiser l'ergonomie, préserver la maintenabilité et la durabilité) et d'efficience (recherche une performance aussi bonne voire meilleure, en consommant moins de ressources).

Oui, et alors ? Voici quelques éléments concrets, par phase de projet :

- Expression du besoin : développer puis exécuter du code qui n'est pas utile et donc très peu utilisé est peu rentable économiquement, et conduit à l'obésité des logiciels. Il sera donc nécessaire lors de l'expression de besoin de bien cerner les besoins de l'utilisateur ;
- Conception de l'ergonomie : une des premières causes de la "fracture numérique" (rupture d'égalité de droit ou d'accès à un service, pour une partie de la population du fait de sa difficulté d'accès à l'informatique) est la complexité d'usage et non pas les problèmes d'accès au matériel. L'accessibilité pour les personnes handicapées sera à prendre en compte mais plus généralement la simplicité de l'interface pour tous, grâce à une phase de conception centrée sur l'expérience utilisateur ;
- Architecture : la répartition des calculs sur différents matériels (clients, serveurs…) aura un impact important sur la consommation de ressources et sur la durabilité du logiciel. La connaissance du nombre d'utilisateurs et de l'usage sera nécessaire ; en effet on privilégiera par exemple des traitements côté serveur pour un logiciel utilisé par de nombreux utilisateurs.
- Choix des technologies : le choix d'un langage ou d'un framework suit de nombreuses contraintes (compétence de l'équipe, socle existant…) ; il est cependant nécessaire de réfléchir à l'impact sur la consommation de ressources mais aussi sur la capacité de la technologie envisagée à perdurer ;
- Développement : la manière dont un logiciel est développé a un impact direct sur le niveau des ressources matérielles nécessaires pour le faire fonctionner. Améliorer un algorithme, ou mieux utiliser une librairie, seront donc bénéfiques pour l'empreinte du logiciel ;
- Tests et mesures : on ne peut pas gérer l'impact du logiciel si on ne mesure pas. Connaître la consommation d'énergie, l'impact mémoire (etc.) sera nécessaire pour prendre de bonnes décisions d'amélioration. Comme pour toutes les anomalies, plus tôt une anomalie énergétique sera détectée, moins sa correction coûtera cher.
- Usage : une fois le logiciel développé, l'utilisateur pourra être sensibilisé aux bonnes pratiques d'usage. Les contributeurs d'un CMS seront par exemple sensibilisés à la réduction de la taille des images qu'ils chargeront dans l'interface.

Et alors concrètement, qu'est-ce que je fais en tant que développeur ? Après toutes les évolutions technologiques, pourquoi optimiser ?

"premature optimization is the root of all evil?"

Un des premiers arguments contre l'écoconception des logiciels (et globalement contre la nécessité d'optimiser le code) est une phrase magique : "**premature optimization is the root of all evil**". Deux mots "premature" et "evil" qui renforcent bien l'idée que penser à l'optimisation du code serait une erreur.

Il est nécessaire de revenir à l'origine de cette phrase avant de la prendre comme un acquis ou une loi de la programmation. Donald Knuth écrit en 1974 :

> "There is no doubt that the grail of efficiency leads to abuse. Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We *should* forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that

> critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only *after* that code has been identified. "

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed"... Peut-être en 1974 mais pas tellement dans les années 2010 ! Nous avons beaucoup de contraintes prises en compte dans les devs (Fonctionnalité, montée en charge, sécurité...) mais pas forcément la performance. La course à l'armement annoncée avec la scalabilité horizontale, verticale, le PaaS, le Cloud en général, les performances toujours meilleures des smartphones... nous ont encore plus fait oublier le besoin d'optimiser. Le message induit par la loi de Moore enfonce le clou. Si on ne comprend pas qu'il n'est pas nécessaire, voir contre-productif d'optimiser, c'est que l'on est passé à côté de pas mal de choses. Alors non, en 2016, les développeurs ne passent plus de temps à penser à l'optimisation.

Ce n'est pas que l'on n'optimise pas les logiciels, on optimise plutôt en "mode pompier". Remontée critique de client, « inutilisabilité » de l'application... On ne voit pas la performance comme un critère d'évaluation mais plutôt la non-performance comme un bug. Et c'est là le problème. Plutôt que de former des armées de développeurs à la culture de la performance, on forme des experts à l'analyse des problèmes de performance. Résultats, des analyses de performance uniquement en période de crise, des logiciels de profiling complexes et peu répandus, et donc coûteux.

Performance vs efficience

Le diable n'est pas dans l'optimisation prématurée et il est dans notre image actuelle de la performance. Quand la performance est intégrée comme une exigence de développement, ce n'est pas forcément des plus idéal. En effet, la performance est une qualité d'usage du logiciel : temps d'affichage, transaction en base... Si l'on prend la norme ISO 25000 sur la qualité, on parle plutôt d'efficience que de performance : [1].

Et quand on regarde ce que cache l'efficience : comportement temporel et utilisation de ressources.

> "resources expended in relation to the accuracy and completeness with which users achieve goals "

On le voit dans le monde Web : la performance Web se focalise sur l'affichage de la page en moins de x secondes. Top ! Quid de l'autre caractéristique qu'est la consommation de ressources ? On oublie l'efficience globale pour se focaliser sur une caractéristique importante mais pas unique. Combien de serveurs utilisés pour l'affichage d'une page ? Combien de requêtes ? Quelle consommation d'énergie ? Et quand on prend en compte le nombre de requêtes, c'est pour uniquement respecter la performance d'affichage. Pourquoi ? Parce qu'un affichage trop long d'une page est vu par l'utilisateur comme un bug, et c'est normal. Mais du point de vue du développeur, le temps d'affichage devrait être vu comme une des caractéristiques d'usage parmi d'autres dans toutes celles couvrant l'efficience du logiciel. Et ce constat pourrait être appliqué à de nombreux axes de l'optimisation logicielle. Les "goulots d'étranglement" des logiciels ont été naturellement gérés : on a optimisé les requêtes coûteuses, on a minimisé les temps de lancement des applications, on va réduire les temps d'affichage des pages...

Mais est-ce qu'on aura travaillé sur l'efficience globale du logiciel ? Non. Web Energy Archive, une étude menée par le Green Code Lab et l'**ADEME** a prouvé qu'il n'y avait aucune corrélation entre la performance Web et l'efficience (et plus particulièrement l'efficience énergétique). Voici la répartition de la mesure sur 500 sites : [2].

On arrive à un logiciel "**uniformément lent**". Pas de ressenti de non-performance gênant pour l'utilisateur parce qu'il accepte la performance actuelle (il ne croit même pas à un temps de lancement immédiat des applications), donc pourquoi aller plus loin ? Ainsi le logiciel devient un peu plus lent :

> "that software is getting slower more rapidly than hardware becomes faster."— https://en.wikipedia.org/wiki/Wirth's_law

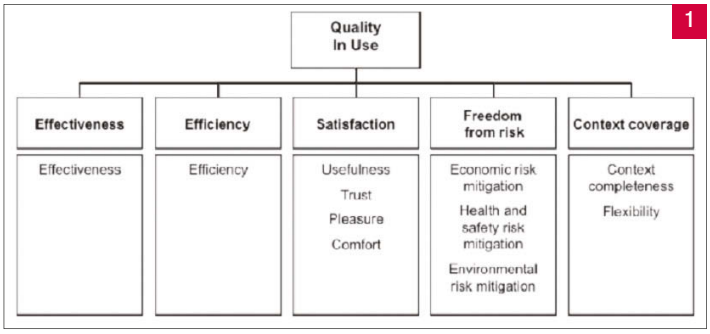
Plein d'arguments si vous ne voulez pas devenir Green Codeur !

Nous trouvons toujours des arguments pour ne pas optimiser :

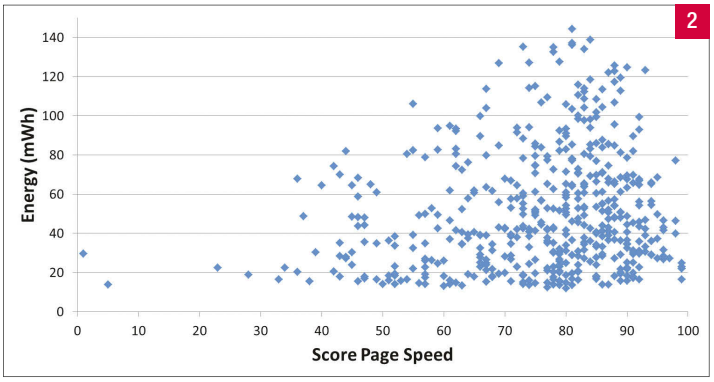
- Optimisation prématurée : « Ouh là, il ne faudrait pas avoir un impact négatif sur la performance. D'autant plus que cela nous demanderait de mesurer la performance, chose que nous ne faisons pas... encore moins pour l'efficience. » ;
- Loi de Moore : « Pourquoi optimiser alors que dans 18 mois, nous aurons des plateformes encore plus performantes ? » ;
- « Le prix du matériel est toujours plus faible ». Effectivement, les prix des serveurs, des VM, de la mémoire ont diminué. OK mais quel est le coût réel d'un site Web sachant qu'en 2 secondes, 40 serveurs sont contactés ? Sur combien de VM sont hébergés les serveurs (on intuite qu'avec la scalabilité, les micro-services..., le chiffre augmente régulièrement) ? ;
- Il y a plus à économiser ailleurs : l'optimisation c'est bien mais c'est pour les autres. Pour les devs, c'est l'infra où il n'y a plus de gras; pour l'infra, ce sont les constructeurs...

Ces freins sont compréhensifs en partie : nous avons été conditionnés pendant plus de 20 ans sur la course à la puissance du matériel et sur l'immatérialité du logiciel. Nous parlons de dette technique dans les équipes de développement ; je pense que la dette technique est encore plus grande. La non-optimisation de l'efficience a créé une dette d'efficience importante. Et l'effort peut être important pour la résoudre : formation des développeurs, méthodologies et outils à mettre en place... Il est plus simple de repousser le problème sous le paillasson ou chez les autres. On le voit dans les GAFA : on parle depuis plusieurs années du Green IT dans les datacenters (free cooling, récupération de chaleur...) mais quid des réelles actions sur la consolidation et l'efficience du logiciel ?

1



2



Les actions existent mais sont encore timides. On voit même des promoteurs du Green IT passer des messages selon lesquels l'éco-conception du logiciel serait un axe technique et mineur. Je dirais majeur et complexe si on devait le traiter sérieusement.

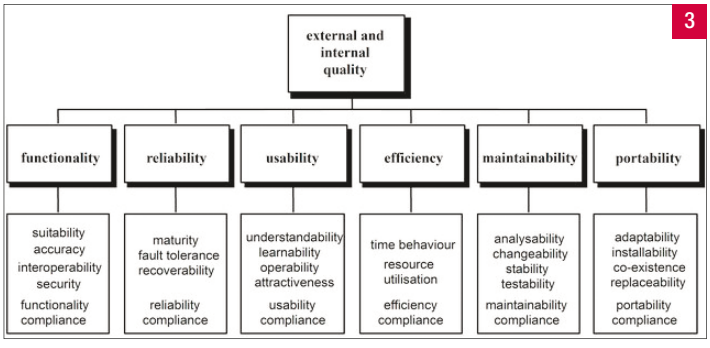
Virtuel, Numérique, Digital, Cloud... Les applications ne sont pas vues comme des objets concrets, comme des produits... Oui le code a une adhérence forte au matériel, oui un code sans matériel ne fournira pas de service. Il est plus simple pour tous d'englober la problématique dans un tout un peu flou qu'est l'IT ou le Numérique. Cependant bien diviser chaque composant, chaque produit permet de travailler sérieusement sur chaque élément. De plus, le logiciel est un produit. Certains essayeront de le nier argumentant qu'il n'existe pas de logiciel sans matériel. Oui un moteur peut être considéré comme un produit, et il n'a pas de sens sans châssis et sans roue, mais c'est un produit. Et on travaille sur son efficience … et ses impacts intrinsèques d'amélioration sur les autres composants. **Cette classification en produit est** aussi nécessaire dans d'autres domaines comme la qualité. De nombreux pays considèrent juridiquement le logiciel comme un produit **(Article un peu vieux mais toujours d'actualité).**

Commodité du développeur vs efficience pour l'utilisateur [3]

De plus, un biais de notre profession n'aide pas à améliorer cela. En tant que développeur, nous avons toujours pour objectif d'améliorer notre productivité. C'est normal, les contraintes devenant de plus en plus fortes : sécurité, cycle de développement plus rapide, évolution des technologies à prendre en compte... Nous prenons donc des choix en fonction de cela. Pour revenir à l'ISO 25000, on privilégie plus la qualité interne du produit qui nous touche que la qualité d'usage. Qualité interne telle que définie par l'ISO : maintenabilité. Maintenabilité, oui, mais il ne s'agit que d'une des 8 caractéristiques de la qualité interne (Portability, Reliability, Compatibility...). Oui c'est important mais il est nécessaire de balancer nos choix et efforts sur toutes les caractéristiques de nos logiciels. La commodité des développeurs ne doit pas être priorisée par rapport aux besoins des utilisateurs. Cela peut paraître une contrainte en plus, mais j'ai confiance en notre capacité d'efficience.

Alors non, optimiser largement le code n'est pas la cause de tous les maux, bien au contraire, c'est une solution. Non, il n'y a pas uniquement 3 % de code critique, mais plutôt 97% de code critique : Frameworks lourds, librairies de plus en plus utilisées et Web services ultra-utilisés. Toutes les expérimentations que je vois le montrent : un petit effort sur n'importe quelle partie du code amène à des diminutions drastiques de la consommation de ressources. La situation de 1974 où l'on était proche du matériel et où le code était uniformément performant et efficient est bien loin. On est maintenant face à un code uniformément lent et non efficient accéléré par de plus en plus de matériel(s). Plus de cœur pour faire tourner un tableur, plus de capacité batterie pour faire tourner nos applications mobiles, plus de datacenters pour répondre à l'IoT. Cela ne serait pas un problème si le coût en ressource et l'impact sur l'environnement n'existaient pas. Mais ce n'est pas le cas. Il est urgent d'oublier les leitmotivs obsolètes de l'industrie logicielle et de ne pas tomber dans la "silver bullet" du Cloud (en tout cas sur l'axe "plus besoin d'optimiser"). Et d'aller au-delà de la seule optimisation de la performance. Et cela vous l'appelez comme vous le voulez : efficience, optimisation de la performance et des ressources, écoconception logicielle, Green Coding...

3



Comment mesurer ?

Vous l'avez compris, il faut d'abord mesurer l'efficience de votre application. Comment ? En mesurant la consommation d'énergie. Les contraintes d'énergie forte ont amené les constructeurs à intégrer des sondes dans le matériel. Le besoin d'avoir un SOC (State of Charge) précis pour les appareils mobiles implique d'avoir des circuits intégrés de mesures sur les plateformes. Cela implique que certaines plateformes mettent à disposition des informations d'énergie. L'avantage de cette approche est d'avoir une mesure intégrée ne nécessitant pas d'ajout d'appareil. Il est cependant nécessaire de s'assurer de la cohérence et de la précision de la mesure. En effet, les mesures peuvent avoir plusieurs limites : précision pas assez importante, fréquence de mesure trop faible, unité de mesure non cohérente…

Note : Pourquoi la mesure CPU ne suffit pas ? La complexité de l'architecture des CPU ainsi que des architectures matérielles (CPU, GPU…) font que le profiling unique du CPU ne suffit pas (et est complexe à analyser). L'énergie est une métrique plus simple et plus globale.

Android et Linux

Le système de fichier linux (et donc Android) intègre les résultats dans le système de fichier. La difficulté est de trouver où sont les informations. Les informations sont généralement dans /sys/. Ensuite cela dépend, il faudra généralement chercher un répertoire contenant battery ou BAT. On trouvera dans ce répertoire plusieurs fichiers avec des noms standardisés (Energy_now, capacity...). Pour un PC Asus Pro :

```
[source,Shell]
----
$ cd /sys/bus/acpi/drivers/battery/PNP0C0A:00/power_supply/BAT0
/sys/bus/acpi/drivers/battery/PNP0C0A:00/power_supply/BAT0
$ ls
alarm          energy_full_design  present       uevent
capacity       energy_now          serial_number voltage_min_design
capacity_level manufacturer        status        voltage_now
cycle_count    model_name          subsystem
device         power               technology
energy_full    power_now           type
----
```

On prendra en fonction du système Energy ou Power. On pourra récupérer les informations :

```
[source,Shell]
----
cat power_now
11477000
----
```

Soit pour la plateforme une puissance consommé de 11,477 Watt.
Sur Android Huawei Mate 8, on trouvera les informations dans :

```
/sys/devices/soc.0/qpnp-smbcharger-ffffffc00ea20400/power_supply/battery
[NOTE]
```

Si vous choisissez de prendre comme unité une puissance, il ne faudra pas oublier de passer à l'énergie en appliquant la formule Energy = Puissance (Watt) * temps entre les mesures (seconde) / 3600

Linux et UPower

Pour Linux, on peut utiliser UPower qui donnera des informations plus structurées :

```
[source,Shell]
----
upower -d
----
```

On aura les résultats suivants :

```
[source,Shell]
----
battery
  present:             yes
  rechargeable:        yes
  state:               discharging
  energy:              72,083 Wh
  energy-empty:        0 Wh
  energy-full:         86,391 Wh
  energy-full-design:  87,001 Wh
  energy-rate:         10,467 W
  voltage:             11,1 V
  time to empty:       6,9 hours
  percentage:          83%
  capacity:            99,2989%
  technology:          lithium-ion
History (rate):
  1463428617   10,467  discharging
  1463428587   11,532  discharging
  1463428557   12,198  discharging
  1463428527   10,733  discharging
----
```

On prendra energy-rate qui est la puissance consommée instantanée.

API Android

Depuis la version 5, Android intègre une API permettant de récupérer les informations d'énergie. Les informations sont disponibles dans android.os.BatteryManager :

- BATTERY_PROPERTY_CAPACITY : Remaining battery capacity as an integer percentage of total capacity (with no fractional part).
- BATTERY_PROPERTY_CHARGE_COUNTER : Battery capacity in microampere-hours, as an integer.
- BATTERY_PROPERTY_CURRENT_AVERAGE : Average battery current in microamperes, as an integer.
- BATTERY_PROPERTY_CURRENT_NOW : Instantaneous battery current in microamperes, as an integer.
- BATTERY_PROPERTY_ENERGY_COUNTER : Battery remaining energy in nanowatt-hours, as a long integer.

```
[source,java]
----
BatteryManager mBatteryManager = (BatteryManager)context.getSystemService
(Context.BATTERY_SERVICE);
return mBatteryManager.getLongProperty(BatteryManager.BATTERY_PROPERTY
_CURRENT_NOW);
----
```

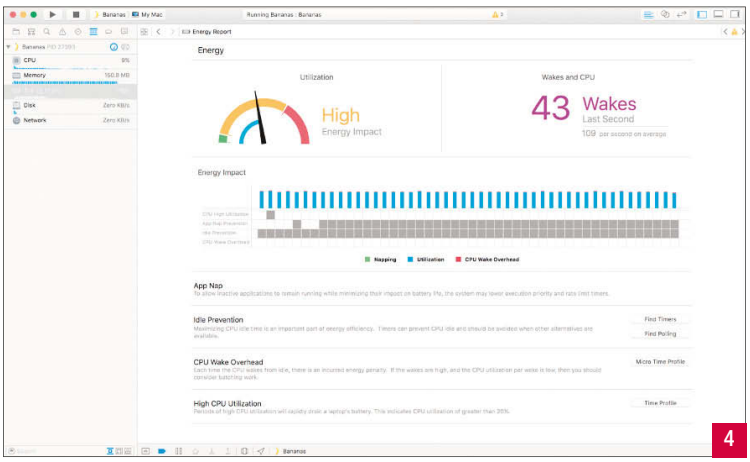
Il faut vérifier toutes les informations. Les constructeurs ne provisionnent en effet pas systématiquement les informations. Il faudra alors utiliser la technique expliquée précédemment : aller chercher les informations dans le système de fichier Linux.

Usage des données

La complexité est d'analyser les données. Mais la solution est la même que pour la performance ou le monitoring des ressources sur serveur : il est nécessaire de mettre en place des tableaux de bord, des rapports, d'automatiser … Pour que l'énergie deviennent un outil de pilotage simple pour tous les développeurs.

iOS

Les outils de développement iOS, comme XCode ou Instrument, implémentent des métriques d'énergie. Cependant elles sont plutôt haut niveau et ne permettent pas d'avoir une finesse pour travailler réellement sur le code. On les utilisera dans un premier niveau de mesure. [4].
Le premier niveau de mesure est disponible directement dans Xcode avec Energy Impact gauge (choose View > Navigators > Show Debug Navigator). Les jauges ne fournissent pas de mesure précise mais un graphique sur la stimulation de la batterie pour l'application qui tourne dans XCode. Une indication globale permet d'identifier si l'impact est élevé ou non. Cette approche a le mérite d'exister mais n'est pas tout le temps applicable car, quand des traitements sont nécessaires, il y aura une indication d'un impact élevé (alors qu'il est peut-être normal). Cependant l'avantage de cette solution est que le profileur fournit des informations détaillées sur la consommation des différents éléments : CPU, Disque et Réseau. Un niveau d'information supplémentaire est possible dans Instrument où il est possible de lancer un diagnostic énergétique. De la même manière que Xcode, Instrument donne une indication sur le niveau d'impact sur la batterie. La métrique est entre 0 et 20. 20 indique que l'autonomie de la



4

batterie est fortement impactée, 0 qu'il y a très peu de sollicitation. On peut regretter de ne pas avoir de valeur plus précise et consolidée. En effet, ce qui est intéressant, c'est l'énergie de la séquence que l'on a exécutée avec l'application. C'est donc l'accumulation de l'impact à chaque seconde. On pourra sur un test par exemple avoir un pic de consommation important mais sur le reste du test un impact faible, la consommation du test sera faible mais on se focalisera à tort sur le pic.

Code dirty / Code Green

Une fois que l'on a récupéré l'énergie dans son code ou dans ses tests, c'est parti : on peut identifier les axes d'amélioration ressource. Voici quelques exemples qui permettent de voir comment intégrer plus de « vert » dans le code. C'est un début, l'écoconception des logiciels étant large (Accessibilité, obsolescence...).

Identifier le budget énergétique des fonctionnalités : cas d'un site Web

Les sites Web sont des très bons révélateurs de ce qui se passe sur l'obésiciel des technologies. Les bonnes pratiques que nous allons voir ici sont transposables à bien d'autres domaines. Ce cas est inspiré de la vie réelle ! L'équipe marketing demande l'intégration d'une timeline Twitter sur la page d'accueil du site. Alors go, on y va. On intègre le plugin Twitter dans le framework Angular. En quelques lignes, on a une fonction intégrée et une librairie en plus sur le site. Dans ce cas, on ressent une perte de performance sur le site. Mais bon, c'est toujours acceptable, d'autant plus que c'est plus beau. Côté requête, c'est l'explosion (image, script…) : [5]

Mais qu'en est-il de la consommation de ressource ? La mesure de consommation de l'énergie nous donne :

L'impact en chargement est élevé (x2 environ) pour l'énergie. Cette notion d'impact est très importante car c'est le surcoût du chargement du site sur la plateforme (OS+matériel). C'est le budget énergétique pour afficher le site avec ses fonctionnalités. L'autonomie s'en ressent : impact de 4 heures sur 11h total si on recharge le site en continu. Et là, on peut discuter avec l'équipe marketing sur la nécessité réelle d'intégration de cette fonctionnalité. [6]

Pour rester dans un budget réaliste, on peut avoir plusieurs solutions :

- Supprimer la timeline ;
- Optimiser la timeline rapidement. On peut en effet configurer la timeline à 2/3 tweets. Il y aura donc moins de requêtes. Les mesures montrent que l'on gagne alors 30 % d'énergie ;
- Ne pas embarquer le plugin Twitter et faire un traitement côté serveur. En effet, la consommation de ressource est principalement issue des multiples requêtes (Au moins 20 requêtes), 5 serveurs contactés…

Cette dernière solution paraît complexe par rapport à "juste" intégrer en quelques lignes le plugin Twitter. Mais c'est bien là la problématique de l'écoconception : on nous a offert de multiples solutions pour nous aider à intégrer des fonctionnalités rapidement, à améliorer notre commodité d'usage des techniques… Mais à quel prix en ressources. La solution est sûrement entre les deux. A nous de trouver des solutions plus efficientes et de ne pas accepter à tout prix toutes les solutions.

5

Simuler l'usage réel de l'utilisateur : cas d'une application Web

Le programmeur « écolo » ne doit pas partir de l'hypothèse que les plateformes matérielles vont continuer à s'améliorer continuellement… Et que même si cela est le cas, tous les utilisateurs ne vont pas avoir accès à ces évolutions. Les utilisateurs n'ont pas tous de la 4G mais parfois de la 2G, beaucoup d'utilisateurs s'équipent en matériel reconditionné et donc n'ont donc pas toute la puissance que les développeurs possèdent… Et nos plateformes à la pointe ne vont pas dans le sens de prendre en compte cela. Avoir une plateforme pour faire tourner un IDE et en même temps être représentatif d'un utilisateur avec une plateforme light est antinomique. Par exemple, pour sensibiliser les développeurs, Facebook a organisé les Mardi de la 2G, et équipé les équipes de téléphones 2G pour faire ainsi comprendre la peine des utilisateurs.

Alors comment faire une application ou site Web écoconçu qui fonctionnerait sur n'importe quelle plateforme et cela avec un niveau d'utilisabilité acceptable ? En optimisant le code, bien sûr, mais surtout en mesurant en environnement réel (ou simulé). Voici un cas sur un portail Web de cartographie (toujours en cas réel) qui explique clairement cela.

Dans un cas de développement, on est en connexion wifi. La consommation d'énergie est importante (143uAh/s contre 75uAh/s en idle) tout comme le temps de chargement de 16s. Cependant compte-tenu de la fonctionnalité, le rendu peut paraître acceptable. [7]

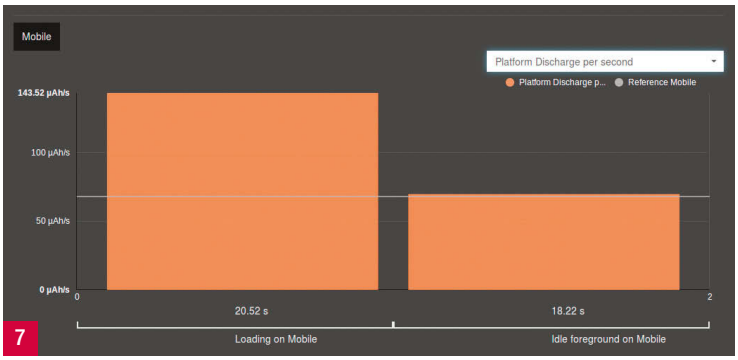
Afin d'étudier le comportement et se projeter sur un usage réel, nous avons effectué les tests sous différentes connexions : Wifi, 4G, 2G(H) et GPRS (E). En 2G, nous avons un temps avec un chargement de 26s.

En simulant la vitesse en 2G, on identifie bien le nombre de connexions importantes mais surtout des temps d'attente causés par 2 scripts majeurs :

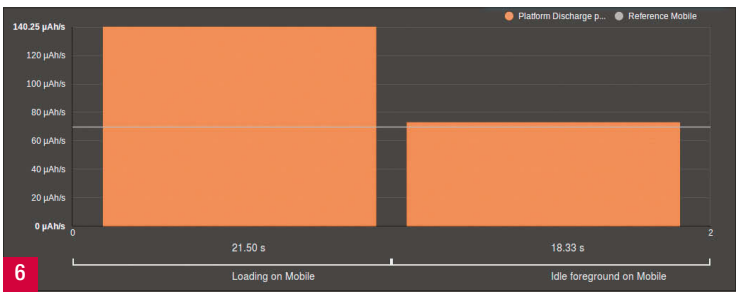
- vendor.js / 282kb / 7,4s de téléchargement en 2G / 760ms en 4G
- Visu-carto.min.js / 446 kB / 8,71 s en 2G / 884 ms en 4G

Le comportement énergétique identifié est expliqué principalement par ces deux scripts issus du framework Ember JS. Globalement, il est ici nécessaire de réduire la taille de ces scripts et de les rendre non bloquants.

Au passage, une fonction très utile sous les outils Google Chrome, le



7



6

Throttling qui permet de simuler le chargement et la performance avec différents niveaux de connexion : [8]
Voici quelques pistes :

- Supprimer les assets non nécessaires https://ember-cli.com/user-guide/#customizing-a-built-in-asset
- Rendre Ember sur le serveur avec Fastboot https://www.ember-fastboot.com/docs/user-guide
- Supprimer les appels synchrones de type XHR comme :

```
J.ajaxTransport(function(e) {
var i = e.xhr();
```

En conclusion, on pourrait dire qu'un code efficient est un code qui fonctionne sur n'importe quelle plateforme et dans n'importe quelle situation d'accès au réseau (Versus qui fonctionne très bien sur ma plateforme ou la plateforme de test).

Le code oui c'est bien, mais quid des images ?
Les images ne sont pas du code mais ont un impact non négligeable sur les sites Web et même les applications mobiles. Par exemple, voici la répartition du poids des éléments sur les pages Web (Source http://httparchive.org/) : [9]
Ce poids impacte les services numériques à plusieurs endroits : les serveurs et CDN (Content Delivery Network) pour servir les images, le réseau, les terminaux.
Pour limiter cet impact, il est nécessaire de réduire la taille des images le plus possible. Plusieurs outils permettent cette optimisation : ImageMagick, pngcrush… Ces outils vont enlever de nombreuses informations non nécessaires (metadata, pixels redondants...). L'intégration de ces outils est d'autant plus simple que des plugins existent pour les chaînes de compilation (Dans Grunt par exemple).
Un gain important peut aussi être obtenu en adaptant l'image à la plateforme de visualisation. Pourquoi envoyer une image retina-ready et non redimensionnée sur un Android avec un écran bas de gamme ? Pour cela, c'est simple on va créer des catégories de définition d'image :

```
<picture>
  <source srcset="image/400px-Olympiastadion_at_dusk.JPG" media="(max-width: 400px)" width=400 height=300>
  <source srcset="image/1195px-Olympiastadion_at_dusk.JPG" width=400 height=300>
  <img src="image/1195px-Olympiastadion_at_dusk.JPG" alt="Flowers" width=400 height=300>
</picture>
```

On peut aller plus loin en adaptant le format d'image. Le jpeg progressive sera par exemple plus intéressant pour les mobiles. Voici une comparaison de la consommation d'énergie en fonction de ces paramètres (Gray : image 1400 redimensionné, Green : image redimensionné avec le src à 400px) : [10]

Comprendre les frameworks : exemple d'Angular

Une fois que l'on a optimisé son architecture, ses fonctionnalités…, que l'on pense que tout va bien, nous avons toujours du « gras » à trouver. Les frameworks actuels sont en effet des consommateurs importants pour nos architectures. Nous avons le temps de monter en compétences sur leur usage mais nous prenons peu de temps pour comprendre leur réel mécanisme de fonctionnement interne. Et cela est dommage car nous pourrions comprendre facilement l'impact sur le matériel.
Angular JS est un très bon exemple de ce qu'un framework peut avoir comme impact. Il permet de faire facilement une application Web riche et dynamique. Cependant les mécanismes peuvent être lourds en ressource. Le binding d'une longue liste avec ng-repeat peut être consommateur car des $watcher vont être attachés à chaque élément. Dans le cas où l'on souhaite afficher des données supplémentaires avec des directives ou des templates (par exemple quand l'utilisateur clique sur la liste), l'usage de ng-show va demander le rendu.
Cela va créer des éléments DOM supplémentaires et l'évaluation de data-binding, et donc aussi du repaint.
Pour afficher des données supplémentaires dans les listes, il est préférable d'utiliser ng-if plutôt que ng-show.
ng-if ne va pas demander de traitement (comme le rendu et le binding).
On évitera de la même manière ng-hide et on préférera ng-switch.

Solution :

```
<li ng-repeat="item in items">
 <p> {{ item.title }} </p>
 <button ng-click="item.showDetails = !item.showDetails">Show details</buttons>
  <div ng-if="item.showDetails">
   {{item.details}}
  </div>
 </li>
```

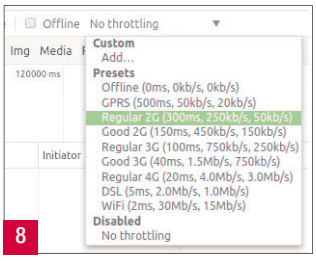
De la même manière, l'ajout d'un filtre rend la liste encore plus consommatrice. En effet, un premier filtrage va être effectué si un objet est modifié et un second pour voir si un nouveau traitement est nécessaire. Il est préférable de filtrer un ng-repeat dans le JavaScript plutôt que dans le HTML. Angular fournit le provider $filter pour cela. Eviter :

```
<li ng-repeat="item in filteredItems()">
```

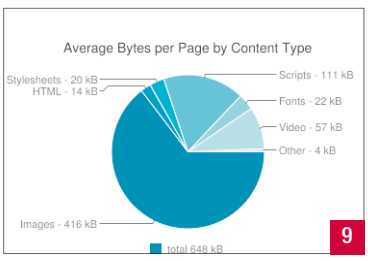
Préférez :

```
<li ng-repeat="item in items">
```

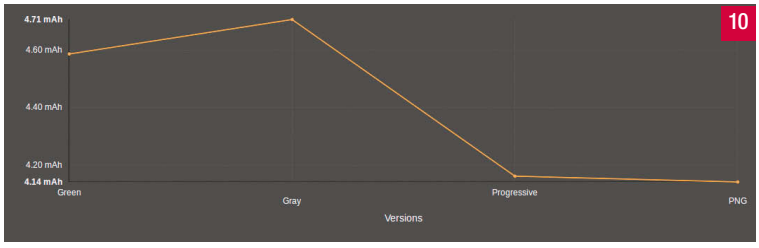
« Connais ton framework, tu sauras comment maîtriser son impact. »



8



9



10

Application Android

L'obégiciel ne se retrouve pas que dans le Web. On retrouve ce phénomène dans tous les domaines et toutes les technologies. Même dans les domaines où les contraintes de ressources sont plus strictes. C'est le cas d'Android.

La mesure de la barre de status Android (notification, état des connexions…) donne les mesures suivantes :

On a donc une consommation qui triple lorsqu'on ouvre et on ferme la barre de status (sur 3 secondes environ). On pourrait se dire que c'est normal, cependant ramené à une fonctionnalité qui est simple (affichage d'icone, mise à jour des icônes et notification), cela semble élevé. D'autant plus que cette action est utilisée de nombreuses fois dans la journée par les utilisateurs. Un profilling Android explique encore mieux cela : [11]

Quelques explications :

- 1er gros pic de traitement : affichage de la quick bar Android ;
- 2ème gros pic : affichage de toute la barre ;
- 3ème gros pic : fermeture de la barre.

Les 8 CPU sont utilisés pleinement pendant le scroll de la fenêtre. Et ce que l'on observe avec les pics est qu'il y a des traitements supplémentaires entre les actions. L'analyse du code Android permet d'identifier que l'event programming est largement utilisé :

- Lors de la première action de l'utilisateur, System UI commence à écouter les nombreux événements Android (Wifi, Batterie, 3G…) ;
- Dès ce moment, les événements sont traités et re-broadcastés en interne par exemple pour le traitement des icônes ;
- Tous les événements sont traités et déclenchent des traitements graphiques (entre autres).

Ce fonctionnement explique les "petits pics" entre les "gros pic". En analysant plus finement, on retrouve aussi ce type de traitement lors des gros pics. Un changement même infime du signal radio va déclencher un redraw de l'icône radio. On arrive à des redraw non nécessaires pour l'utilisateur.

L'ajout de tests permet de limiter le nombre de redraw :

```
@Override
public void setIsAirplaneMode(IconState icon) {
  // We have to check if the airplane mode has changed
  if (mInfo.airplaneModeEnabled == icon.visible) return;
  mInfo.airplaneModeEnabled = icon.visible;
  refreshState(mInfo);
}
```

[12]

On diminue comme cela de 4% la décharge batterie mais surtout on évite des traitements non nécessaires… et on laisse les ressources disponibles pour d'autres applications.

Alors pourquoi un tel fonctionnement arrive-t-il dans du code provenant directement de Google (qui pousse et applique des bonnes pratiques !) ? On retrouve ici l'effet rebond des nouvelles pratiques et technologies : évitez le pooling et passez à l'event programming ! Très bien, cependant pas de silver bullet, il faut contrôler les évènements. Peut-être un orchestrateur dans l'application permettrait-t-il une meilleure gestion de tous ces événements ? Sinon l'event programming va devenir le nouveau pooling programming ! •

NAVIGATEURS, LANGAGES, MIDDLEWARES… QUELLE EST LA PART DE CONSOMMATION ?

Le développeur responsable ne rentrera pas dans une guerre trollesque : le langage vert c'est l'assembleur, php c'est trop lourd, C c'est mieux que Java… Chacun des langages et frameworks ont leurs avantages. Cependant, ils ont aussi leurs coûts en ressource. L'idéal avant de se lancer dans un projet est de réfléchir à la technologie : quel est le besoin final, quel est le coût en ressources, est-ce qu'une autre technologie ne sera pas préférable en termes d'efficience ? Ce n'est pas simple mais cela permet de débuter avec une dette environnementale avant de débuter le projet. Pour le mobile : natif vs hybride vs site responsive vs progressive Web app ? Avec de l'hybride, je vais couvrir plus facilement un panel d'appareils et rester sur ma technologie de prédilection, cependant le « hello world » va déjà me créer ma dette environnementale (plus de 100Mo de mémoire avant de commencer à coder). De l'autre côté, une application native sera plus optimisée mais aura un potentiel d'obsolescence avec l'évolution des API Android (par exemple). Pour les applications Web, il faudra prendre en compte l'impact du navigateur qui est nécessaire.
La notion de budget énergétique d'un framework ou d'une technologie est importante dans ce cas car elle va permettre de faire des choix critiques en début de projet .

Le boite à outils du Green Codeur

Voici quelques pistes pour débuter dans le Green Coding.

Outils de profilling

Tous les outils de profiling classiques sont utiles. La limite est qu'aucune information d'énergie n'est disponible et qu'ils sont généralement complexes.

- Google Dev tools : en particulier les onglets Network et Timeline qui donneront toutes les informations de consommation de ressources ;
- Outils Android Traceview et Systrace : Traceview est un outil qui va permettre de voir graphiquement le comportement de l'application et sa performance. Systrace permettra d'avoir plus d'information sur les interactions avec le système et plus particulièrement le système graphique ;
- Linux Perf tool : de multiples possibilités d'analyser la performance du système et de votre application : https://perf.wiki.kernel.org ;
- Les outils intégrés dans les IDE : tous les IDE intègrent des outils de profiling.

Outils spécifiques éco-conception

Encore trop peu nombreux mais utiles car plus dédiés à l'énergie et l'écoconception !

- Power API : sonde de l'INRIA qui permet d'estimer l'énergie à partir d'information système ;
- Web Energy Archive : mesurez la consommation d'énergie de vos sites et récupérez une étiquette énergétique !
- Greenspector : suite logicielle permettant de travailler sur l'écoconception (mesure d'énergie et analyse de code) ;
- Battery Historian ; outil Android permettant de récupérer les informations batterie et d'application.

Les Silvers Bullets du codage vert

Pour éviter de croire qu'il y a une solution magique pour rendre nos logiciels virtuels et comment ne pas tomber dans le piège (et vraiment utiliser l'avantage en faveur de l'efficience).

Silver Bullet	Explication de l'argument	En quoi cela pourrait être une silver bullet	Pourquoi cela ne risque de ne pas être une silver bullet
Cloud	Ne t'occupe plus des ressources consommées, juste du code, ton fournisseur s'en occupera !	Oui car la mutualisation et la bonne gestion par le fournisseur va amener une efficience	Ton code a un impact certain sur le framework SaaS, sur la VM et sur le matériel. Et le fournisseur n'y peut rien
Open source	Open donc green et efficient (Combat Windows vs Linux)	L'open source et plus particulièrement Linux est plus efficient : architecture modulaire et possibilité de travailler sur l'efficience	Un logiciel est un logiciel, privateur ou libre; il subit les mêmes règles de l'obésité.
Open Hardware	L'ouverture des plateformes, le DIY… va résoudre les pb d'obsolescence	Nous allons reprendre la main sur les plateformes et pouvoir mieux gérer l'adéquation au matériel	Nous allons avoir les mêmes problèmes que sur le matériel fermé (réinvention de la roue !).
Nouvelles architectures plus efficientes	Evolutions, donc amélioration de l'efficience	Les nouvelles architectures vont prendre en compte les erreurs du passé et s'améliorer	Il y a systématiquement un effet rebond ('voir exemple Android)

Les compétences 2017 : notre horoscope

Judicaël Paquet
DSI chez Batiwiz, Judicaël est spécialiste dans le coaching agile et l'architecture logicielle/devops

L'informatique évolue à grande vitesse et il est très intéressant de suivre les grandes tendances du marché. Nous allons voir les profils qui sont les plus demandés pour cette année 2017 dans le monde de l'informatique.

L'agilité et le devops accélèrent

Les entreprises accélèrent de plus en plus leur transformation en optant pour les méthodes agiles. Quelques grands groupes tentent malgré la difficulté d'adopter le Scrum. Cela implique de changer son organisation et de trouver des profils adaptés à ces méthodologies de travail ; certaines entreprises en font même un critère d'embauche pour les développeurs : ouvert à l'esprit d'équipe, ayant des connaissances de la culture Agile… Cette année, de nombreuses entreprises recherchent des Product Owners et des Scrum Masters, deux rôles clés du Scrum. Nous voyons également une belle accélération de la culture Devops : la moitié des entreprises se disent prêtent à y passer dès 2017 ; 9,4% des DSI disent en faire leur préoccupation principale. Si ce n'est pas encore le profil le plus demandé à ce jour, le profil devops est cependant de plus en plus recherché et sera probablement difficile à trouver dès le courant de l'année 2017. On voit d'ailleurs de nombreuses formations fleurir sur Internet pour proposer aux administrateurs systèmes de devenir des devops.

Java n'est pas mort

Nous entendons régulièrement (et surtout dans les communautés des autres langages) que le Java va mourir. C'est plutôt le contraire car le Java est redevenu le langage le plus recherché par les entreprises depuis 2 ans, et les demandes ne font qu'augmenter depuis plusieurs années. D'ailleurs, on verra après que les profils Android font partie de ces profils très recherchés en Java. Dans l'expertise Java, il y a toujours les profils Hybris (CMS ecommerce) qui sont encore demandés bien que la demande semble diminuer ces derniers temps. Nul doute que le Java reste et restera le langage le plus populaire de l'année 2017.

Javascript et ses frameworks très populaires

La communauté est très active et nous propose d'ailleurs plusieurs framework de qualité dans le domaine ; Nodejs et ses frameworks ont permis au Javascript de devenir le deuxième langage le plus populaire du marché. Ces profils que l'on appelle développeurs front-end (full stack pour ceux qui font aussi du back-end) sont les profils qui ont selon moi les plus évolués (salaires en vive augmentation, profils enfin considérés comme des développeurs et non des intégrateurs). Si le Javascript est populaire, les recherches de profils sont principalement sur deux frameworks des plus populaires du moment : Angular JS et React JS. Les profils NodeJS sont également très recherchés mais moins que les deux frameworks précédents. Si d'autres frameworks émergeaient comme Backbone ou Meteor il y a quelques années, ils ne sont plus du tout demandés sur le marché actuel. Jquery qui était le framework le plus popuplaire pour ne pas dire le seul sur le marché il y a 10 ans, a totalement disparu. Beaucoup d'applications ou de sites l'ont encore mais il ne fait plus du tout partie des plans de refonte des entreprises. Si un développeur Javascript ne se lance pas en AngularJS ou en ReactJs, je crains qu'il finisse par devenir un profil obsolète en 2017. Si on associe le Javascript au front-end, il faut savoir que le nombre d'entreprises qui optent pour avoir du back-end en Javascript sont de plus en plus nombreuses. Certains grands groupes font d'ailleurs partie de ces entreprises.

Et PHP au milieu de tout ça ?

Le développeur PHP qui était encore le profil le plus recherché il y a 2 ans, semble en perdition. Ne nous alarmons pas, il reste très recherché mais il est vrai que le langage semble vivre un tournant ces dernières années. En revanche, une tendance qui ne change pas mais qui est impressionnante sur le marché est de voir des recruteurs qui recherchent des développeurs Symfony (on ne parle même plus de développeurs PHP). On voit d'ailleurs cette tendance se confirmer avec le dernier Drupal où les recruteurs recherchent des développeurs Drupal. C'est d'ailleurs assez paradoxal mais malgré la mauvaise progression du PHP ces dernières années, ces deux types de profils sont les plus difficiles à trouver dans le monde du développement à ce jour.
J'insiste sur ces deux profils car les recruteurs confirment qu'ils concernent plus de 70% des demandes. Les développeurs PHP qui n'ont pas d'expérience dans ce domaine ont beaucoup plus de mal à se vendre aujourd'hui même si le marché tendu actuel leur reste favorable pour trouver un emploi. Alors qu'on voyait Laravel arriver sur le marché français, il semble vivre une progression beaucoup moins forte que ce qu'on imaginait. Cependant vu sa popularité dans d'autres pays, nul doute qu'il continue à progresser sur le marché français. Par contre, période difficile pour le framework Zend qui se fait de plus en plus rare sur le marché. Il y a encore quelques demandes de profils Zend mais il est vrai que ce n'est plus trop demandé.

Le mobile est aussi très actif

De nombreux acteurs misent de plus en plus sur le mobile qui est devenu un canal indispensable de communication ou de vente. Les profils développeurs IOS et Android sont de plus en plus recherchés. Les demandes ne sont pas aussi nombreuses que dans les langages cités auparavant mais il y a beaucoup moins de développeurs dans ce domaine. Les recruteurs semblent en effet avoir beaucoup de difficultés à trouver leurs profils. Si les outils de cross-plateforme semblaient prendre de l'ampleur pendant des années, la tendance est actuellement aux applications natives. Il est évident que si le mobile vous tente, c'est probablement une bonne année pour vous lancer car la croissance de l'utilisation d'un mobile (dont pour l'acte d'achat) est chaque année très importante.

Déjà de belles annonces pour 2017

Les développeurs devraient vivre encore une très belle année surtout quand on voit les annonces faites par de grands groupes comme par exemple Vente Privée qui recherche 250 personnes dès début 2017 et principalement des développeurs.

Et les autres langages ?

J'ai pris le temps de parler des langages les plus populaires mais il y a de la demande dans d'autres langages. Le C#, C++ et Python sont des langages qui sont encore recherchés à ce jour ; mais il est vrai que leur marché semble moins tendu que ceux cités précédemment. Beaucoup moins populaires ces dernières années, le Ruby, le Perl et le C sont néanmoins toujours des profils sur le marché, mais il est vrai que ces langages attirent de moins en moins les jeunes recrues.

La pénurie du Data Scientist

Il est vrai que je n'en parle que tardivement dans cet article mais ce profil encore peu connu en France est l'un des plus difficiles à trouver. Pas étonnant que la tendance continue cette année car pour information c'est le poste le plus recherché (par rapport au nombre de profils disponibles) aux Etats Unis. Dans le monde du Big Data, ce poste est l'évolution du Data Analyst. Si vous avez une bonne capacité d'analyse, que vous avez des connaissances dans un outil analytique tel que SAS ou R, une maîtrise d'un langage comme Python, Perl ou C++, des notions de machine learning, une maîtrise de SQL et potentiellement de Hadoop, alors vous pouvez vous lancer dans ce type de poste très intéressant. Je pense que la pénurie de profil que vit ce secteur peut vous permettre de vous proposer en tant que profil junior.

Le développeur BI

Le Big Data continue sa progression et les entreprises recherchent également des développeurs BI. Ces profils seront toujours recherchés lors de l'année 2017 vu la tendance de progression du Big Data ces dernières années.

Côté chef de projet

De ce côté les entreprises recherchent également des postes de MOE et d'AMOA (certains associent ce poste au Product Owner). Pour ce qui est des chefs de projet technique, il est vrai que la tendance Agile semble effacer peu à peu ce type de profil. Beaucoup de chefs de projets techniques migrent d'ailleurs vers des postes de Scrum Master.

CONCLUSION

Comme vous l'aurez compris, les profils Symfony et les profils Data Scientists sont des profils très demandés en cette période. Cela devrait perdurer tout au long de l'année 2017 pour les profils Symfony et s'amplifier pour les data scientists. Les profils Javascript qui sont de plus en plus demandés devraient également vivre une continuelle progression avec l'utilisation de plus en plus massive du Javascript en back-end. Les profils Java sont également des profils très demandés et cette année 2017 ne devrait pas voir la tendance s'inverser. Les autres profils que j'ai cités sont évidemment toujours demandés sur le marché mais il est vrai qu'ils ne vivent pas la même dynamique. Vous connaissez à présent les tendances du marché, donc vous saurez vers où aller si vous avez envie de changer pour cette année 2017. Vous avez toutes les clés en main pour bien gérer ce changement. •

LES COMPÉTENCES 2017 : PRÉDICTION DE LA RÉDACTION

Il est toujours difficile de prédire les compétences ou profils qui vont émerger, se confirmer ou fléchir. Voici notre résumé pour cette année.

• François Tonic *Programmez!*

Plus que jamais, le développeur doit regarder autour de lui, suivre les conférences, lire Programmez!, se former toute l'année. Un profil technique se travaille toute l'année et sur la durée. Le marché reste tendu sur certains profils. Les profils "généralistes" avec 0 à 2 ans d'expérience, tels que développeur Web, développeur Java ou C#, se trouvent relativement facilement dans les régions et villes actives.

Sur les salaires, nous ne voyons pas beaucoup d'évolutions, surtout, dans les profils 0 à 2 ans d'expérience et "généralistes" et même dans le jeu vidéo où la concurrence est féroce entre développeurs. Certaines industries cherchent des profils très spécifiques comme C++ embarqué, Scala ou la programmation fonctionnelle. Peu d'annonces mais aussi peu de candidats. Les différences entre l'Île de France et la province restent valables. Cependant, les grandes métropoles ou régions très actives sont aussi intéressantes que Paris et sa région, par exemple : Grenoble, Nantes, Bordeaux, Toulouse, Lille, Montpellier, Marseille. Cela ne signifie pas que le reste de la France ne soit pas interessant, bien au contraire, mais là les salaires peuvent être plus bas que la moyenne. •

Technologie / langage	Tendance	Commentaire
Développeur Web	- à +	Développeur Web veut tout dire et rien dire. Aujourd'hui, ce sont des profils plus précis qui sont recherchés comme sur les CMS ou JavaScript. Un dév Web généraliste risque de stagner. Proposez des compétences spécifiques.
CMS	+	Aucun risque, une bonne expertise Drupal continuera à être reconnue.
PHP	= / +	Là encore, une expertise PHP sera appréciée surtout si vous la couplez avec une autre compétence.
Python	+	Pour nous, c'est le langage à surveiller.
C#, Java, C++, C	=	Nous nous entendons à de grosses surprises sur ces langages. Ils continueront à être demandés, notamment Java à cause du lourd patrimoine en production. Cependant, pour sortir du lot, notamment avec les récents diplômés, ou nouveaux formés, n'hésitez pas à acquérir des compétences supplémentaires sur des outils et les évolutions les plus récentes.
Cloud Computing	+	Le développeur Cloud émerge encore peu en France sauf à faire uniquement de l'Amazon Web Services ou du Salesforce. Soyez proactif sur le sujet.
Mobilité	+	La mobilité reste très forte et la demande ne baisse pas. Attention tout de même, beaucoup de profils recherchés sont des profils ayant des compétences Android et iOS (Objective-C, Swift), la maîtrise d'outils de Xamarin, Cordova, Unity est vivement conseillée. Vous pouvez être développeur mono-plateforme mais dans ce cas, ayez une expertise en béton.
JavaScript et les frameworksJS, Angular	+	Même si vous détestez JS ou Angular, ce sont deux compétences importantes à ne pas négliger surtout si vous voulez vous former ou changer votre profil de développeur Web.
Sécurité, DevOps, agilité	= / +	Ces compétences vous aideront dans le quotidien, mieux vaut les connaître que de les ignorer.
Docker, conteneurs	+	On en parle beaucoup mais pour le moment, ce n'est pas un profil différenciant. Cependant, ne négligez pas en 2017 les conteneurs et formez-vous dans les prochains mois.
Développeur full-stack	?	Full-stack ou pas ? Ce profil complet se voit souvent sur les annonces mais attention à la définition très large que l'on peut lire. Prudence.
Open Source	+	Certains profils sont de facto open source. Si en plus, vous êtes actif en communauté et dans les conférences, c'est un + indéniable et à négocier avec l'entreprise.

L'**obsolescence** des compétences : informaticiens, attention pour votre carrière

Cyril PIERRE de GEYER
Cyril Pierre de Geyer dirige les Executive MBA du groupe Ionis (EPITA, Epitech, ISG...), est professeur lié à HEC et gère la société Openska (www.openska.com), spécialisée dans le conseil et la formation Web, data et open source. Il a mis en pratique son expertise PHP pour de nombreuses références importantes telles que BlaBlaCar, Unesco, Drupagora, Orange, France Télévisions…

C'est une donnée encore peu connue. Les études à son sujet sont rares. Et pourtant, l'obsolescence des compétences est un phénomène qui s'impose dans la vie professionnelle d'un actif, particulièrement chez les informaticiens.

Négligée, elle peut avoir des conséquences fatales sur sa carrière.
Appréhendée, elle offre des perspectives d'épanouissement et d'évolutions.

Durant le travail, le fait d'estimer que ses compétences sont hors d'usage n'est pas réellement une nouveauté. Seulement hier, cela pouvait se manifester à quelques heures de la retraite. Aujourd'hui, tout va vite, très vite. L'émergence de nouvelles technologies n'a de cesse d'accélérer le jeu. Dorénavant, l'obsolescence des compétences se rapproche après quelques années passées en entreprise. Elle peut survenir suite à une évolution technologique, suite à une évolution professionnelle ou simplement à cause d'un changement de mode.

Course contre la montre dans le numérique ?

Dans notre secteur, en informatique, l'obsolescence est d'autant plus caractérisée que l'évolution technologique est exacerbée. Certaines approches évoluent presque tous les six mois. La façon de créer un site Web, aujourd'hui, est complètement différente, comparée à il y a seulement quelques années. Et que dire des méthodes de communication. Il y a un an, on parlait à peine d'Angular, c'est l'un des sujets les plus demandés chez Openska à l'heure actuelle. Ce sont autant de compétences disparues aux profits d'autres.

Focus évolution des langages

Parlons d'informatique. Si l'on pense en termes de programmation, une limite à l'obsolescence des compétences est qu'il existe deux paradigmes dont découlent la majorité des langages : la programmation procédurale et la programmation objet. Connaître l'un ou l'autre, voire les deux, permettra une prise en main rapide d'un nouveau langage basé sur le même paradigme.

Sortir de sa zone de confort

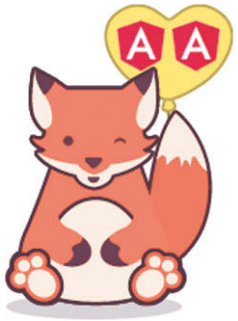
Plus on développe son expertise, plus on se spécialise et plus c'est difficile de passer à une autre technologie.
Pourquoi ? D'une part par confort, on est très compétent sur un sujet, on est très sollicité dessus et repartir sur une zone ou votre expertise peut être remise en cause demande beaucoup de courage. D'autre part de façon totalement pragmatique une entreprise préfèrera positionner cher un expert sur une technologie plutôt que de le positionner moins cher sur une autre technologie.
Pour autant il est vital pour l'informaticien d'évoluer et de ne pas se risquer trop longtemps sur une technologie/approche unique. Les risques sont multiples mais le pire est peut-être d'être amené à ne plus évoluer, de faire toujours la même chose…

La formation pour évoluer

Un levier, et non des moindres pour se tenir à jour, la formation professionnelle, qui enrichit de façon pragmatique et ciblée son panel de compétences. Elle remet très vite les pendules à l'heure et permet de rester dans la course avec de nouvelles qualifications.
C'est un must que tout professionnel doit considérer tout au long de sa vie. Pour Houily DU, manager chez Openska « les demandes en termes de formations sont un bon observatoire des tendances. »

Les tendances : top 10 des demandes de formation chez Openska en 2016

1	Angular
2	PHP 7
3	JavaScript
4	Docker
5	Symfony 3
6	Drupal
7	Talend
8	PHP Expert
9	QlikView
10	SCRUM



La confirmation de l'arrivée massive de JavaScript dans le jeu se confirme, le framework Angular est à l'honneur, suivi de loin par son alter égo ReactJS (11ème position). Côté back-office la sortie de PHP 7 avec son boost de performances et ses nouvelles fonctionnalités montrent que PHP n'est pas mort, loin de là. La mouvance devops se matérialise avec des demandes sur Docker et sur les méthodes agiles, notamment Scrum.
Consciente et maîtrisée cette obsolescence offre des perspectives, à savoir d'une part, l'enrichissement de son bagage professionnel. Et d'autre part, elle facilite une pluralité de l'activité tout au long d'une carrière. Encore faut-il prendre le temps de lever la tête du guidon pour observer et comprendre les changements. Une caractéristique fondamentale : la curiosité. •

JavaScript : des origines à NodeJS Partie 1

• Teddy DESMAS
• Guillaume LEBORGNE
• Thomas OUVRE
Développeurs chez **Infinite Square**
http://blogs.infinitesquare.com/

JavaScript ou JS est un langage de programmation Web basé comme son nom l'indique sur des scripts. Il a été créé en 1995 par Brendan Eich, un informaticien américain travaillant alors chez Netscape en s'inspirant de Java (d'où son nom) mais en simplifiant la syntaxe de ce dernier pour faciliter son utilisation notamment pour les débutants. JavaScript fut standardisé 2 ans après, en juin 1997 sous le nom d'ECMASCRIPT. Il est alors devenu une implémentation de ce standard par la fondation Mozilla, et certaines entreprises comme Microsoft ou Adobe ont implémenté leur propre version (respectivement JScript et ActionScript).

Ce langage a permis de rendre dynamique les sites Web en les modelant et en manipulant le code HTML. C'est devenu aujourd'hui un poids lourd du développement. Nous en entendons de plus en plus parler, et ce, partout ! Aussi bien par les développeurs front (qui sont la cible première), mais également par les développeurs mobiles, back et ce bien sûr au niveau du marché de l'emploi. Les développeurs JavaScript sont très recherchés par de nombreuses entreprises, petites et grandes, et cela peu importe le marché. Mais pourquoi un tel engouement ? Pour les développeurs, cet engouement est certainement dû à plusieurs facteurs. D'une part grâce à sa communauté qui est très grande et très active, car elle propose de nombreux outils, aides, plugins qui facilitent grandement l'accès à ce langage. D'autre part, avec ses nombreux avantages techniques qui le rendent plus dynamique, plus flexible comparé à d'autres langages du marché. JavaScript est en premier lieu un langage fonctionnel, mais il permet aussi une approche objet car c'est un langage prototypé (on peut faire de l'héritage et créer des instances comme avec un langage objet). C'est donc un langage extrêmement flexible. Cependant, JavaScript reste un langage de script.

C'est donc un langage interprété, et qui ne bénéficie pas d'un typage statique, et donc pas d'étape de compilation pour vérifier la cohérence du code. Pour pallier cela, il existe des outils et frameworks qui permettent de vérifier son code (ESLint, JSLint,…) ou d'ajouter du typage statique (comme TypeScript et Flow). Par ailleurs, le JavaScript est compris par tous les navigateurs, mais pas forcément au même niveau. Les scripts s'exécutent côté client et sont très dépendants de l'appareil du client. Il faut donc faire très attention à la compatibilité du code qu'on écrit pour que celui-ci puisse fonctionner sur tous les appareils ciblés, sous peine d'avoir des ralentissements ou des scripts qui tombent en erreur.

De plus en plus, la flexibilité et la portabilité du JavaScript sont vus comme des avantages, notamment pour les entreprises. Un autre avantage et non des moindres, est que le JavaScript possède une communauté vraiment dynamique. On trouve de nombreuses librairies pour répondre aux problèmes courants. Les entreprises évoluent également et on entend de plus en plus parler d'environnement full JS, dans lesquels on utilise aussi le JavaScript pour les parties serveur. Des avantages cités pour les entreprises en découlent donc un avantage fort pour les développeurs JavaScript ; ils sont très recherchés par les entreprises et n'ont pas cet effet de niche que peuvent avoir certains langages de par l'utilisation minime qui en est faite dans les entreprises, ou bien par rapport aux faibles secteurs qu'ils peuvent adresser. A première vue, JavaScript semble donc être un leader et un langage fort, mais comment cela se traduit-il dans le temps. JavaScript est-il toujours en évolution, ou commence-t-il à stagner ?

Son évolution et le futur

Et depuis qu'il a été créé, il faut dire que JavaScript a beaucoup évolué. Il dispose actuellement de nombreuses fonctionnalités permettant des réalisations impensables lors de ses premiers balbutiements.

La liste des évolutions est tellement longue qu'il est difficile de savoir par quoi commencer. Nous allons volontairement faire l'impasse sur celles qui sont purement liées au code et celles que nous ne jugeons pas impactantes. Nous allons donc nous concentrer sur les fonctionnalités, outils et frameworks qui ont permis à JavaScript de réaliser une grande plus-value, comme par exemple l'ouverture vers un nouveau marché, ou bien l'augmentation significative de la productivité des développeurs. Ces différents points vont être détaillés de manière succincte dans cette partie, mais certains seront plus détaillés dans la suite de cet article.

Serveur

En effet, dans un premier temps cantonné au rôle de code client, JavaScript s'est vu doté d'un nouvel atout, celui de code serveur. Il est en effet possible de réaliser un site Web et de l'exécuter côté serveur uniquement avec du JavaScript, et ce avec Node.JS. Cette plateforme a été créée par Ryan Dahl et vous permet d'exécuter votre application Web écrite en JavaScript. Bien que très récente, elle est extrêmement prisée par la communauté, notamment par sa robustesse, sa scalabilité native et par ses performances qui sont très attractives. C'est pourquoi de nombreuses entreprises ont déjà utilisé Node.JS dans leurs activités, notamment Microsoft, PayPal, Netflix, LinkedIn et bien d'autres encore.

Mobile

A l'époque où nous sommes actuellement, le mobile tient une place très importante, et les besoins en applications mobiles sont extrêmement forts. Les développeurs JavaScript l'ont bien compris ! Le Web étant une technologie commune à toute les plateformes mobiles (iOS, Android, Windows Phone/Mobile), JavaScript est le langage idéal pour faire du développement cross-plateformes avec le moins de disparités. C'est ainsi que plusieurs Frameworks dédiés à ce type de besoin, ont fait leur apparition. Le plus connu est sans aucun doute Apache Cordova, développé par la fondation Apache, qui permet aux développeurs JavaScript de réaliser des applications riches, en ayant à leur disposition de nombreux composants et fonctionnalités des plateformes. Cordova met à disposition des plugins (réalisés par la communauté) pour accéder à des fonctionnalités natives depuis le code JavaScript. Cordova est une approche dite hybride car elle permet d'encapsuler un site ou une application Web sous la forme d'une application. Il y a donc un compromis sur la fluidité, l'accès aux res-

sources de l'appareil, et l'expérience utilisateur par rapport à des applications natives. Cependant, beaucoup d'entreprises font le choix d'utiliser cette approche, au détriment du développement natif. Notamment car elle permet de gagner beaucoup de temps de développement (1 code pour N plateformes). De plus, dans de nombreux cas, le Framework permet de réaliser tout ou partie des fonctionnalités que l'application cible (dans de rares cas, certaines fonctionnalités ne sont encore disponible qu'en natif, et le développement d'un plugin natif est nécessaire). Ces dernières années, d'autres outils reposant sur JavaScript permettent de réaliser des applications mobiles. Contrairement à Cordova, ces nouvelles plateformes permettent de réaliser des applications natives. C'est le cas par exemple de React Native et de NativeScript. Dans ces approches le rendu de l'application repose sur les composants natifs, JavaScript n'étant utilisé que pour le code de l'application.

Framework

Il existe notamment JQuery, qui reste extrêmement populaire et parfois confondu avec JavaScript lui-même. Il permet de modifier le DOM de manière simplifiée, de réaliser des animations, d'accéder à de nombreux évènements … Angular et React ne sont pas non plus en reste et sont de plus en plus mis en place lorsqu'il faut réaliser des applications riches. Le premier est un Framework développé par Google qui met en avant l'architecture MVC (Modèle-Vue-Controller) connue par les développeurs et qui encourage un couplage faible entre les différentes couches de l'application, à cela s'ajoute le data-binding qui permet une synchronisation automatique des modèles et des vues. Le second, conçu par Facebook est une bibliothèque qui ne gère que l'interface. Elle peut être utilisée en la couplant avec d'autres bibliothèques ou même avec Angular. Son approche est une approche en mode composant, c'est-à-dire un bloc dans une page représente un ou plusieurs éléments, ce qui permet une réutilisation des composants et une meilleure séparation des différentes parties qui composent une page. De plus, en travaillant avec un DOM virtuel et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité, React permet de réaliser des applications Web très performantes. De nombreux autres Framework existent comme Backbone, Ember, Polymer, et ont chacun leurs fonctionnements différents et leurs avantages, inconvénients.

Test

JavaScript se doit d'être équipé côté test. Pour ce faire, de nombreux Frameworks et outils de tests ont fait leur apparition. De Mocha à Casper en passant par Karma et VorlonJS, ils permettent de réaliser de nombreux tests sur le code JavaScript pour en assurer le bon fonctionnement. Sans les tests, une application n'est pas maintenable et il devient très difficile pour le ou les développeurs en charge de celle-ci de s'assurer de son bon fonctionnement après l'ajout de fonctionnalité ou la correction de bug. Avec toutes ces évolutions et toutes les facettes que peut adresser JavaScript, on peut se demander : jusqu'où va-t-il aller ? Avec d'un côté, les évolutions matérielles, les nouveaux périphériques (IOT, …), les nouveaux protocoles, les nouveaux standards, et de l'autre, sa croissance, sa communauté, JavaScript risque de devenir encore plus présent. Les Frameworks ne cessent d'évoluer proposant de nouvelles fonctionnalités qui enthousiasment fortement les développeurs et donc les entreprises en leurs donnant une envie constante d'utiliser les dernières nouveautés. Chose que les autres langages ont parfois beaucoup de mal à réaliser.
Nous allons maintenant rentrer dans le vif du sujet, avec comme vous l'attendez tous, des explications techniques et des exemples de codes avec ECMAScript, Babel et TypeScript.

ECMAScript

JavaScript évolue, en suivant notamment les normes ECMAScript. On compte à ce jour 3 normes ECMAScript en cours d'adoption, à commencer par la 6 (ECMAScript 2015). Elle apporte son lot de nouveautés pour le langage et permet aux développeurs de faire appel à des concepts jusqu'ici absents comme les classes (dont on pouvait toutefois émuler le fonctionnement), les lambdas, ou même le couple de mots clefs très populaires en C# async/await (avec ES7). Néanmoins, le JavaScript étant un langage interprété, ce qui signifie que son code est lu et exécuté sur son environnement d'exécution, on peut se demander comment nos navigateurs qui ne recevront pas de mise à jour de leur moteur JavaScript pourront fonctionner avec ces nouvelles syntaxes et fonctionnalités. C'est là qu'interviennent les transpilers JavaScript.

Babel : un transpiler ES6 vers ES5

La version de JavaScript fonctionnant quasiment universellement se base sur ES5. Babel est une forme de compilateur traduisant du code JavaScript ES6 vers du code JavaScript ES5. Comme il s'agit d'une compilation traduisant du code d'un langage vers un autre, on nomme ce procédé transpilation.

Getting Started

C LI

Le site de Babel fournit toute une batterie d'exemples pour en commencer l'usage au sein de nos projets, que ce soit avec la CLI Babel, un système de build comme Browserify, Grunt, Gulp, Webpack et autres, ou même dans un programme C# ou Ruby.
Commençons donc par installer la CLI via NPM :

```
npm install --save-dev babel-cli
npm install babel-preset-latest --save-dev
```

Nous avons aussi besoin d'un preset, ici le dernier, qui permet de configurer Babel de sorte à compiler certaines fonctionnalités (en l'occurrence toutes celles prises en charge dans Babel, pour les versions 2015,2016 et 2017 d'ECMAScript). Il est ensuite possible d'utiliser la CLI comme suit :

```
babel main.js --presets es2015 --out-file bundle.js
```

Faites précéder la commande de « .\node_modules\.bin\ » selon la présence ou non de l'exécutable Babel dans votre path, ou de l'exécution de cette commande via NPM. Ainsi, le fichier main.js :

```
class Hello {
    world() {
        console.log("hello world");
    }
}
```

Devient, une fois compilé :

```
"use strict";
var _createClass = function () { /*SKIPPED*/ }
function _classCallCheck(instance, Constructor) { /*SKIPED*/ }
var Hello = function () {
    function Hello() {
        _classCallCheck(this, Hello);
    }

    _createClass(Hello, [{
```

```
    key: "world",
    value: function world() {
      console.log("hello world");
    }
  }]);

  return Hello;
}();
```

Toutefois, l'exemple précédent se basera sur le fait que la fonction require est définie, ce qui est le cas si l'on travaille en nodeJS par exemple, mais pas si l'on souhaite exécuter notre fichier dans un navigateur. Plusieurs solutions existent, mais si l'on souhaite un système recommandé en production, il faudra faire appel à un outil tiers pour la compilation, comme Webpack, Browserify ou même Gulp. Ainsi pour installer Browserify et le plugin nécessaire, on installera les 2 packages NPM suivant :

```
npm install --save-dev browserify
npm install --save-dev babelify
```

On créera ensuite un fichier de configuration Babel que l'on nommera « .babelrc » :

```
{
  "presets": ["es2015"]
}
```

La ligne de commande suivante permettra ensuite de fournir un fichier « bundle.js » contenant l'ensemble des dépendances au fichier d'entrée « main.js », que l'on pourra inclure dans un fichier HTML par exemple :

```
browserify main.js -t babelify --outfile bundle.js
```

Note : *attention, cette commande s'exécute normalement via NPM (définie donc le package.json), preffixez la de « .\node_modules\.bin\ » pour un test direct.*

Polyfills

Ces nouvelles versions de JavaScript n'introduisent pas seulement de nouvelles syntaxes de code, mais aussi de nouvelles APIs. Il est donc nécessaire de faire appel à une librairie de polyfills pour émuler le fonctionnement de ces APIs si elles ne sont pas prises en charge nativement. Babel offre bien évidemment une réponse simple à cette problématique sous la forme d'un fichier JavaScript à inclure dans nos projets. Il est possible de l'obtenir via NPM :

```
npm install --save-dev babel-polyfill
```

Le fichier dist/polyfill.js, peut être référencé dans un fichier HTML si l'application s'exécute dans un navigateur par exemple, mais le package peut aussi participer à une compilation Webpack ou Browserify en incluant le package avec l'une des deux méthodes suivantes :

```
require("babel-polyfill");
import "babel-polyfill";
```

L'usage de ce polyfill assure donc la présence de certaines fonctionnalités dites « built-in » ou intégrées à l'environnement d'exécution, dont suivra une liste non exhaustive.

ECMAScript 2015

Dans un projet correctement configuré selon son environnement d'exécution cible, il est donc tout à fait possible d'utiliser les fonctionnalités à suivre, qu'elles soient syntaxiques ou atRuntime, même si votre cible ne supporte que le JavaScript selon la norme ES5.

Classes

L'usage du mot clef classe est maintenant possible en JavaScript. Il apporte un peu de sucre syntaxique à la programmation prototypée et permet de définir plus simplement un comportement orienté objet. L'héritage ainsi que les méthodes statiques sont évidemment aussi supportés :

```
class aBaseClass {
  constructor() {
    console.log("ctor aBaseClass");
  }
}

class aClass extends aBaseClass {
  constructor() {
    super();
    console.log("ctor aClass");
  }
  static factory() {
    return new aClass();
  }
}

aClass.factory();
// ctor aBaseClass
// ctor aClass
```

Syntaxe plus concise pour les objets littéraux

Jusque maintenant, définir un objet littéral se faisait à l'aide d'une syntaxe proche du JSON, du style « key : value ». Comme pour les classes, il est maintenant possible de faire appel à une notation plus proche de celles utilisées pour les classes :

```
var aObj = {
  aMethod() {
    // do something
  }
}

var methodName = "another_method";
var anotherObj = {
  aObj,
  [methodName]() {
    // do something else
  }
}

console.log(typeof aObj.aMethod); // function
console.log(anotherObj.aObj === aObj); // true
console.log(typeof anotherObj.another_method); // function
```

Symboles

Il s'agit d'un nouveau type permettant d'indexer au sein d'un objet, alors que cela ne pouvait être fait qu'avec une chaîne de caractères sous ES5. On pourrait imaginer ce genre de scénario :

```
let aKey = Symbol("description");
class aClass {
  constructor() {
    this[aKey] = "data";
  }
```

```
}

var obj = new aClass();
console.log(obj["description"]); // undefined
console.log(obj[aKey]); // data
```

Note : *attention, pour définir un nouveau symbole, il ne faut pas utiliser le mot clef new.*

Il est ainsi possible de définir les clefs des membres de nos objets de façon plus libre, sans se soucier d'une éventuelle collision avec un membre d'une autre librairie. En effet, le symbole créé est unique. Cela a permis notamment de créer un symbole définissant la fonction itératrice d'une classe, comme nous pourrons le voir ensuite.

Boucle For-of (et itérateurs)

Qui n'a pas déjà tenté, au moins une fois, d'utiliser le couple for-in sur un tableau en JavaScript pour se rendre compte qu'il ne s'agit pas exactement d'un foreach habituel ? Le couple for-of vient donner une nouvelle façon d'itérer sur les objets, et ne donne plus une référence sur la clef de la propriété comme avec le for-in, mais sur la valeur comme on peut parfois s'y attendre. Ainsi l'exemple suivant est fonctionnel :

```
for(var key in ["a", "b", "c"]) {
  console.log(key) // 0 1 2
}
```

Mais retourne les valeurs 0, 1 et 2. Alors que le suivant :

```
for(var item of ["a", "b", "c"]) {
  console.log(item) // a b c
}
```

Retourne les valeurs a, b et c comme prévu. Comme pour les langages C#, Java ou C++, ES6 apporte les itérateurs, soit des objets exposant une fonction retournant un élément d'une collection. La syntaxe for-of se base sur les itérateurs. Ce principe permet entre autres de laisser le soin à celui qui implémente une collection d'en gérer l'itération, au contraire d'une classique boucle for. Pour autoriser l'itération au sein d'une classe ou d'un objet, on y définira un membre à l'aide du symbole Symbol.iterator :

```
class aClass {
  constructor(max) {
    this._max = max;
  }
  [Symbol.iterator]() {
    // must return an iterator
    var ix = 0;
    var max = this._max;
    return {
      next() { // next function is required for an iterator
        return {
          done: ix >= max,
          value: ix++
        }
      }
    }
  }
}
////
for(var value of new aClass(5)) {
  console.log(value); // 0 1 2 3 4
}
```

Les générateurs

Les générateurs simplifient l'implémentation des itérateurs en permettant l'usage du mot clef yield (comme en C#). Ainsi l'exemple précédent pourrait être écrit ainsi :

```
class aClass {
  constructor(max) {
    this._max = max;
  }
  *[Symbol.iterator]() {
    // must return an iterator
    for(var i = 0; i < this._max; i++) {
      yield i;
    }
  }
}
```

Notez l'usage du caractère « * » sur la définition de la méthode, autorisant l'usage du mot clef yield. Il est aussi possible d'utiliser la notation au sein d'un objet littéral :

```
var max = 5;
var test = {
  [Symbol.iterator]: function* () {
    for (var i = 0; i < 5; i++) {
      yield i;
    }
  }
}
for (var value of test) {
  console.log(value);
}
```

Interpolation de string

Il est maintenant possible de créer des chaînes de caractères avec une syntaxe plus agréable lorsqu'il s'agit d'en concaténer plusieurs. Ainsi, l'exemple suivant affiche vrai :

```
var name = "ES6";
var s1 = "programming with " + name + " is cool!";
var s2 = `programming with ${name} is cool!`;
console.log(s1 === s2); // true
```

Il est donc possible d'inclure des variables à l'aide du caractère « $ » et des accolades dans une chaîne ouverte et fermée à l'aide du caractère « ` » (Alt Gr + 7 sur un clavier Azerty). Un autre intérêt de ce type de chaîne est qu'il permet d'incorporer des sauts de lignes (qui seront conservés), ainsi le code suivant est lui aussi valide :

```
console.log(`a string with
a break`);
Et équivalent cette fois-ci au code ES5 suivant :
console.log("a string with\r\na break");
```

Paramètres

La nouvelle norme apporte aussi les paramètres optionnels d'une fonction, dont on peut donc définir une valeur par défaut si celle-ci n'est pas spécifiée par l'appelant :

```
function optionals(p1, pOpt2 = "default", pOpt3 = null) {
   console.log(p1, pOpt2, pOpt3);
}
////
optionals(1); // 1 default null
optionals(1, 2); // 1 2 null
optionals(1, 2, 3); // 1 2 3
```

Il est aussi possible d'utiliser la notion de « rest parameters » qui peut être traduite littéralement par « reste des paramètres ». Cette fonctionnalité indique qu'un paramètre préfixé par « … » (toujours le dernier paramètre de la fonction) doit contenir sous la forme d'un tableau la liste des paramètres restant fournis à la fonction :

```
function rest(p1, ...list) {
   console.log(list);
}

rest(null, 1, 2, 3, null, "hello"); // null [1, 2, 3, null, "hello"]
```

Ce type de syntaxe indique donc clairement qu'une fonction peut recevoir un nombre d'arguments indéfini, alors qu'auparavant nous pouvions atteindre le même résultat de façon moins évidente, en manipulant l'objet « arguments ». L'opérateur « … » permet aussi de passer chaque élément d'un tableau en argument d'une fonction :

```
function spread(p1,p2,p3) {
   console.log(p1,p2,p3);
}

spread(...[1,2,3]); // 1 2 3
```

Déconstruction (destructuring)

Cette fonctionnalité donne accès à une syntaxe comparable à du « pattern matching » et offre entre autres la possibilité de définir des variables pointant vers une cellule d'un tableau ou un membre d'un objet :

```
var array = [1, 2, 3];
var [a, b, c] = array;
console.log(a, b, c); // 1 2 3
```

Les variables a, b et c sont automatiquement assignées selon leur position à la valeur correspondante du tableau. Le procédé est similaire avec des objets :

```
var object = { p1: 3, p2: 4 };
var { p1: d, p2: e } = object;
console.log(d, e); // 3 4
```

Un cas un peu plus complexe :

```
function hello({a, b, c = 3, d: { e, f: [g, , h, ...i]}}) {
   console.log(a, b, c, e, g, h, i);
}
hello({ a: 1, d: { e: 4, f: [5, 6, 7, 8, 9] } }); // 1 undefined 3 4 5 7 [8,9]
```

On remarque la possibilité de définir des valeurs par défaut (sur « c »), d'ignorer une cellule d'un tableau (entre « g » et « h ») et même de récupérer le reste des valeurs d'un tableau (sur « i »).

Fonctions fléchées (lambdas)

Les fonctions fléchées (syntaxe aussi connue sous le nom de lambda) offrent une écriture plus concise pour définir une variable avec une fonction. Ainsi la notation suivante :

```
var myFunc = function(p1, p2) { /**/ };
```

devient :

```
var myFunc = (p1, p2) => { /**/ };
```

La syntaxe peut devenir plus légère encore dans le cas où l'on souhaite définir une fonction avec un seul paramètre ainsi que dans le cas où l'on n'y exécute qu'une seule assertion :

```
var myFunc = function(p1) { return p1 === undefined; };
```

devient :

```
var myFunc = p1 => p1 === undefined;
```

Collections

On peut voir apparaître 4 nouveaux types de collections avec ES6 : « Set », « Map », « WeakSet » et « WeakMap ». Un « Set » est une collection itérable de valeurs uniques :

```
var set = new Set();
set.add(1);
console.log(set.size); // 1
set.delete(1)
console.log(set.size); // 0
```

Une « Map » est une collection itérable elle aussi, de valeurs indexées selon une clef unique (semblable à un dictionnaire) :

```
var map = new Map();
map.set('a', 0);
map.set('b', 1);
console.log(map.has('a')); // true
console.log(map.has('c')); // false
console.log(map.size); // 2
```

Une version dont le nom est précédé de « Weak » existe pour ces deux types de collections. La différence entre ces deux versions, bien que le fonctionnement reste similaire, est assez importante : la première (la version non « weak ») est itérable et tient une référence aux objets qu'elle contient, alors que la seconde (version « weak ») non. Cela signifie qu'un WeakSet n'empêchera pas le garbage collector de libérer la mémoire d'un object qu'il contient si celui-ci n'a plus aucune référence active. La WeakMap opère de la même façon pour les clefs. En revanche, il est impossible d'obtenir la taille d'une telle collection, ou même d'en parcourir les valeurs. Seule la méthode « has » est valide et permet de savoir si une valeur appartient ou non à la collection.

Proxies

Les proxies sont une fonctionnalité de ES6 qui de par sa nature ne peut être émulée via un polyfill. Il faut donc que l'environnement cible supporte nativement celle-ci pour pouvoir en bénéficier. Elle donne ainsi accès à un jeux d'APIs permettant de tracer programmatiquement parlant les opérations effectuées à partir d'un objet, comme l'accès à l'une de ces propriétés, l'appel d'une fonction ou d'un constructeur, l'ajout ou la suppression d'une propriété, etc.

```
var myobj = {
   f1: function() {
      console.log("f1 called");
   }
};
```

```
var proxy = new Proxy(myobj, {
   get: function(receiver, name) {
      console.log(`getting ${name}`);
      return typeof receiver[name] === "function"
         ? function(...args) {
            console.log(`calling ${name}`);
            return receiver[name].apply(receiver, ...args);
         } : receiver[name];
   }
});

proxy.f1();
// getting f1
// calling f1
// f1 called
```

On peut donc facilement imaginer des scénarios de log ou de profiling par exemple.

Let et const

ES6 introduit 2 nouveaux mot clefs pour la déclaration de variables en plus de « var » : « let » et « const ». Le premier permet de spécifier une portée restreinte au scope courant :

```
var v1 = 0;
{
   var v1 = 1;
}
console.log(v1); // 1

let v2 = 0;
{
   let v2 = 1;
}
console.log(v2); // 0
```

Le mot clef const quant à lui restreint à une unique assignation sur une variable :

```
const v3 = 2; // ok
v3 = 1; // compiler error
```

Les modules

Les modules sont un moyen de créer des composants au sein d'une application. Un fichier va pouvoir servir de module, et d'autres fichiers pourront le référencer à travers une syntaxe propre à ES6. Dans un fichier nommé « modules_demo.js » :

```
export default function f1() {}
export function f2() {}
export let v1 = 1;
```

Il est ainsi possible de faire appel au module nommé « modules_demo » dans un autre fichier de la sorte :

```
import * as d_modules_demo from "./modules_demo";
d_modules_demo.f1(); // runtime error : f1 is not defined
d_modules_demo.f2();
let v1 = d_modules_demo.v1;
```

Ainsi la totalité des membres exposés via le mot clef « export » dans le module peuvent être utilisés via une variable que l'on nommera comme on le souhaite (ici « d_modules_demo »). On peut aussi choisir au cas par cas les membres à exporter :

```
import {f2} from "./modules_demo";
f2();
```

Il est aussi possible d'obtenir le membre exporté par défaut avec cette notation :

```
import ff1 from "./modules_demo";
ff1();
```

Les deux précédentes notations peuvent être combinées :

```
Import ff1,{f2} from "./modules_demo";
```

Les modules, avec le bon outillage offrent des fonctionnalités avancées comme le chargement à la demande. Dans un cas plus classique, ils permettent, lors de la compilation, de détecter toutes les dépendances du fichier que l'on considère comme le point d'entrée, et de les inclure dans la sortie.

Promises

Avec le temps nous avons vu apparaître un certain nombre de patterns et techniques pour la gestion du code asynchrone, toutefois les promises (ou promesses) se sont largement répandues, notamment à travers des librairies et frameworks très populaires comme AngularJS, JQuery, etc. ES6 normalise l'implémentation des promises. Ainsi, une fonction devant attendre pour retourner un résultat, peut de façon classique prendre en paramètre un callback à appeler une fois ce résultat obtenu, ou bien retourner une promise. Cette dernière se traduit par un objet exposant au minimum une fonction « then ». Celle-ci peut être vue comme un moyen de s'abonner au changement de statut du traitement asynchrone : on pourra être notifié lorsque le traitement est terminé avec succès ou échec, ou même pendant sa progression.

```
function waitAsync(time) {
   return new Promise((resolve, reject) => {
      setTimeout(function() {
         resolve();
      }, time)
   });
}
waitAsync(1000).then(() => {
   // called after 1 sec
   return waitAsync(500);
}).then(() => {
   // called after 1,5 sec
   throw "error";
}).catch((reason) => {
   console.log(reason); // error
});
```

Comme on peut le voir dans cet exemple, s'abonner via « then » sur une promise permet non seulement d'attendre que le traitement asynchrone soit terminé, mais aussi de chaîner les promises entre elles en retournant une nouvelle dans la fonction donnée à « then ». L'appel à catch permettra de gérer une erreur se produisant dans la promise. Une façon de lever une erreur lorsque l'on définit le corps de la promise consiste aussi à appeler la méthode nommée ici « reject ». On retiendra aussi qu'une alternative au « catch » consiste à passer un deuxième paramètre à la fonction then, une autre fonction dont le but sera de gérer les erreurs.

La suite dans Programmez! 205

Histoire & futur de l'informatique

Épisode 2

• Sylvain Abélard
(@abelar_s)
Architecte logiciel
Faveod

Les principes de l'innovation sont toujours les mêmes : une société, une économie se posent des problèmes. On imagine, applique, croise et assemble des siècles d'idées et techniques. Dans la première partie, nous avons vu comment sciences et techniques, numérisation et automatisation ont pris une place grandissante, sur quelles bases et dans quelles directions.
Mais cela nous laisse avec un état de l'art complexe et difficile à démêler, couplé à des problèmes de mode, d'égo et de politique qui ne facilitent pas la tâche.

HIER
UN ÉTAT DE L'ART
Les outils du métier

Nous disposons d'une pléthore d'outils, dont certains se remplacent, se complètent, ou se ressemblent un peu trop, et font alors l'objet de débats sans fin : on croit que tel outil est remis en cause par l'autre partie, alors qu'il propose d'améliorer telle autre partie de notre métier.

Pour coder

Le plus évident est qu'un outil doit répondre à un besoin.
L'équipe de développeurs fabrique son produit doit donc vivre dans un contexte où les demandes arrivent (style et méthode de management et d'expression du besoin), produire du code (langage, bibliothèque, IDE) et cela en équipe (outils de gestion de projet et de partage/versioning de source, comme svn ou Git), puis vérifier son travail avant de livrer (tests unitaires ou d'intégration, automatisés ou non, écrits avant le développement ou non).
Et pour l'instant, les améliorations constatées ne semblent que marginales : on code toujours dans des fichiers texte. Certes, les IDE proposent de lancer les tâches communes au clic d'un bouton ; de rechercher, naviguer et même proposer une complétion automatique. Mais est-ce là la limite de ce qu'on était en droit d'attendre de plusieurs décennies de travail et d'innovation ?

Pour fonctionner

L'application développée doit ensuite être envoyée (de manière plus ou moins automatique) vers les terminaux (unique à très nombreux) qui l'exécuteront (matériel et plateforme d'exécution), et souvent pratiqueront une analyse plus ou moins poussée de son fonctionnement : logs, sécurité (SIEM - Système de gestion de la sécurité de l'information) et performance.
Inutile de faire une liste : ces contextes sont très différents dans le cas d'un jeu sur console, d'un outil industriel, d'un objet physique (GPS, montre), ou d'une application Web.
Cependant, cela vaut le coup de se poser la question, car les idées sont communes, et chaque innovation ou changement est à reconsidérer dans le périmètre de chacun pour voir si elle vaut le coup.
On peut parler de déploiements rares ou fréquents, sur une flotte réduite ou très large, et sur des machines que l'on possède ou non : contrôlerez-vous deux ans après, pour appliquer un correctif de sécurité, un objet connecté ou la version de l'application mobile de vos clients ?

Pour la durée

Tout ce que l'on peut souhaiter à un projet informatique, c'est de durer dans le temps : il apporte une valeur continue (ou profite des habitudes) et perdure là où il est déployé.
Personne n'est éternel, et cela signifie non seulement une gestion d'équipe adaptée, une formation continue, une documentation à la fois claire et concise... Cet arbitrage n'est jamais simple, et, qui plus est, les développeurs ne sont pas connus pour leur goût à rédiger de la documentation.
Il faut donc s'imaginer devoir, dans le futur et à un moment inconnu, répondre à une urgence ou une demande d'évolution d'un produit historique.
Les tests aideront les équipes à condition qu'elles soient toujours là et mobilisables. La documentation aidera, à condition d'être à jour. Sans cela, l'éternelle question reviendra : recommencer à zéro ou ajouter toujours plus de modifications bancales au socle du produit ?
Là encore, bien que l'industrie ne l'ait pas encore adoptée en masse, plusieurs solutions existent pour la génération de documentation à partir des différents artefacts disponibles : spécifications, code...
Enfin, nous avons la chance de travailler sur des octets, peu coûteux à multiplier (ça n'a pas toujours été vrai) ; et par rapport à une masse d'appareils physiques, il sera toujours techniquement possible (mais peut-être difficile) de faire fonctionner une version de l'application dans un bac à sable, et tenter d'en pratiquer la rétro-ingénierie : deviner les specs, l'architecture, le code et pourquoi pas améliorer et relancer des tests à partir de tout livrable existant, ne serait-ce qu'une application sans même le code source.

Pour la sécurité

Ces techniques sont là aussi connues et souvent dotées d'outils pour assister les personnes qui le font. En ce moment cela est le plus souvent réalisé par des profils qui s'intéressent à la sécurité, qui doivent tester les vulnérabilités d'une application en n'y ayant que des accès très restreints, de manière à voir si elle présenterait des failles à un assaillant.
Mais encore une fois, à quel prix ? Vos entreprises, équipes et projets ont-ils prévu des plans pour la fin de vie du logiciel, une passation de connaissance, une formation continue, des audits fréquents, un plan de continuité et de reprise d'activité en cas d'incident ?

L'informatique c'est cyclique

Fort de cette histoire très riche, nous nous retrouvons avec un présent tout aussi riche et compliqué. Passé et futur cohabitent.
On a le "legacy", l'historique toujours en place avec lequel il faut composer.
On a les expérimentations, dont le futur dira si elles marchent ou pas, et encore : ce qui fonctionne dans un certain contexte n'est pas forcément ce qu'il faudra répéter dans d'autre contextes, et il est tout à fait valable de reprendre ce qui a échoué pour des raisons de contexte plus tard, pour le remettre au goût du jour, parce que les présupposés ont changé.
Et on a le, ou plutôt les états de l'art, perpétuel-

lement en phases d'expansion/diversification et de consolidation/standardisation.

Le monde du développeur

Abstractions et limitations

Lors de la conception des premiers langages, les paradigmes proposés ont pour but d'améliorer la puissance d'expression, la concision du code, et de limiter les erreurs. Le but est simple et humble : gérer toutes ces abstractions est complexe, il faut que tout puisse tenir dans un cerveau humain, qui ne se focalisera que sur une poignée de choses à la fois.

Le but ultime : que le résultat de ce travail abstrait, un énorme mille-feuille de couches empilées qui s'impactent les unes les autres, soit plus facile à naviguer, à expliquer, à présenter sous formes de schémas plus ou moins détaillés en fonction de l'angle selon lequel on approche la chose.

Par exemple, l'innovation du "scope lexical" : on peut considérer qu'il est plus simple que toutes les variables soient globales (modifier nombre_clients dans une fonction change cette valeur partout dans l'application), mais si c'est très facile à concevoir, c'est aussi très facile de causer des erreurs par inadvertance, surtout dans des programmes à la complexité croissante.

On va alors considérer que la variable, le champ, la méthode a un certain nom sur une certaine portée (au sein d'une certaine classe, d'une même fonction, etc), qui n'a rien à voir avec un élément de même nom disponible ailleurs (autre fonction, autre bloc if...).

Paradigme et mysticisme

De nombreuses approches sont alors prévues en admettant que travailler dans un monde abstrait est complexe et qu'il est facile de s'y tromper. Chacun proposera ses astuces pour éviter ces étourderies et cadrer le développeur : programmation objet, orientée aspect (OOP/AOP), développement guidé par les tests ou par les comportements (TDD/BDD), organisation des bibliothèques, des frameworks, de l'infrastructure, architectures N-tiers ou microservices... On finit par vouloir mélanger des choses qui ont un réel impact, mais ne sont pas franchement comparables.

Chacun son héritage et sa manière de voir les problèmes, les découper, et de composer des solutions à l'aide de plusieurs briques. Là encore il est tentant, mais difficile et dangereux de vouloir comparer ces paradigmes dans l'absolu : un besoin aura une réponse plus naturelle dans telle vision ou dans telle autre.

Bref, il y a des habitudes, mais pas vraiment d'incompatibilité entre la programmation objet et la programmation fonctionnelle par exemple : bien que l'on parle de "langages objet" pour Java, C++ et C#, et de "langages fonctionnels" pour OCaML ou Haskell, la plupart des langages fonctionnels sont aussi orientés objet.

Bien entendu, c'est une nuance qu'il est facile d'oublier. Comme toujours, moins un point est important plus il est débattu : les "guerres de langages" entre développeurs, à moins d'arguments prouvés sur votre contexte, sont donc probablement à éviter, ou à voir comme un passe-temps distrayant dans lequel chacun va pousser sa méthode actuelle qui ne changera pas ses habitudes.

Mais la nouveauté ne vient par définition que par les gens qui ne sont pas satisfaits par l'état actuel. Heureusement, certains préfèrent se retrousser les manches, implémenter et tester, que de débattre longuement. Devant le nombre d'idées mises à disposition et d'outils que l'on prend l'habitude d'utiliser, les approches finissent par converger, et on voit arriver des langages qui choisissent dans ce menu les idées qui leur conviennent pour les consolider.

Complexification et rigidification

Plus le temps passe et plus les succès s'alignent, plus on tente de soumettre des problèmes à l'approche informatique.

Bien qu'ils viennent de secteurs complètement différents (énergie, logistique, marketing, vente), on retrouve régulièrement des similitudes : les problématiques sont subies, étudiées, catégorisées, et des solutions innovantes proposées.

Tant que l'informatique est une "science" nouvelle, industrie et académie se renvoient la balle, les experts passent d'un monde à l'autre : physiciens, statisticiens, linguistes apportent à la jeune science des ordinateurs ses lettres de noblesse ; industries et services embauchent pour solutionner leurs problèmes ou faire croître leurs entreprises.

Mais plus le temps passe et plus la spécialisation intervient : là encore, ce qui ne peut pas tenir dans un seul cerveau, ou une seule équipe, finira par se faire découper, plus ou moins rationnellement.

Cela permet d'aller de plus en plus loin dans des équipes d'experts, mais limite également le partage des connaissances de chaque état de l'art.

Global et transversal

Certaines de ces améliorations sont purement abstraites (comme la métaprogrammation, pour pouvoir inspecter et reprogrammer le logiciel lui-même durant son exécution), d'autres purement techniques (la récupération de mémoire, l'optimisation de code), d'autres encore purement pragmatiques et basées sur l'expérience (les patrons de conception d'architectures complexes).

D'autres transforment en un seul coup plusieurs parties de cette chaîne de création : la génération de code, par exemple, touche à la fois à la conception, l'implémentation, la performance et la sécurité, tout en proposant une modification de la façon dont les équipes fonctionnent, de l'expression de besoin à la réalisation et au maintien en conditions opérationnelles. Mais qui aujourd'hui se permet d'avoir une vision aussi transverse ? Qui a l'autorité en entreprise de modifier autant de processus qui affectent autant de directions ?

Le monde du manager

Gestion, organisation

En parallèle à ces philosophies et approches, à côté de ces langages et outils, il est bon de rappeler que le monde technique vit pour et par les métiers.

Les organisations connaissent elles aussi cette évolution naturelle : après une phase d'expérimentation qui ouvre des portes, il faut rationaliser et intégrer les nouveaux process.

Pour certains, c'est la boîte de Pandore : difficile d'encadrer cette IT que l'on ne comprend pas, et dont les coûts et impératifs transpirent pourtant jusque dans la stratégie business, tout comme la stratégie business finit par impacter l'IT au grand désespoir des développeurs.

Juste retour des choses : les métiers sont devenus IT, maintenant l'IT doit (re)devenir métier !

Communication, réunification

Dans les outils mis en place, le plus évident semble être d'adopter un langage commun. Là encore les manières d'approcher et répondre au problème peuvent être radicalement opposées : notations, outils uniques, façons de travailler ou méthode sur étagère ?

UML (Unified Modeling Language), BPMN (Business Process Model and Notation) répondent à ce besoin par une notation, et de nombreuses normes successives. Des suites logicielles entières peuvent se baser sur ces standards, et générer une partie du code et de la documentation attendus.

Le conseil est bien connu : laissez les meilleurs faire des outils que le reste de l'équipe utilisera, réduisant les risques et améliorant la productivité. Mais l'effet Tour d'Ivoire n'est pas loin... Ces

profils sont-ils les meilleurs pour "vendre" les outils qu'ils conçoivent aux équipes de développement, aux métiers et aux acheteurs ?

Standardisation, calcification

Encore une fois, l'industrie repère les zones répétitives, l'inefficience, et propose une solution automatique et intégrée, au prix de davantage d'abstraction. Mais les approches, la complexité et les connaissances à avoir, font que les compétences sont d'autant plus rares et chères, ce que les process achats ne peuvent comprendre au moment où toute l'industrie se délocalise.
Vu la diversité d'outils, une réponse peut être de choisir un ERP (Enterprise Resource Planning) et de le conserver. On est alors rapidement prisonnier d'un vendeur qui peut modifier ses tarifs à loisir (c'est le retour au "customer lock-in" de l'âge du hardware) et surtout de sa manière à lui de voir votre métier : si tout le secteur a le même problème et la même solution, où est votre valeur ajoutée ?
Si le vendeur ne s'est pas penché sur votre problème, ou s'il a des contraintes uniques par rapport au secteur, que faire ? Entrer votre métier de force dans les cases de l'ERP, ou sortir du cadre de l'ERP et tout re-développer ?

Rassurer, certifier

Les échecs s'accumulent, les délais s'allongent, la fracture se fait plus visible entre phase de découverte et de production.
On peut y répondre par un cycle en V ou une méthode "waterfall" : les besoins "descendent" pour produire une direction, des spécifications générales puis détaillées, une phase d'implémentation, et "remontent" via des tests unitaires puis d'acceptation, à la livraison finale.
C'est une méthode naturelle pour limiter les risques, qui limite aussi la flexibilité. Partout où il faut accepter les changements, les équipes acceptent qu'un logiciel n'est jamais "fini" mais en constante évolution.
Le divorce est consommé lorsqu'une réunion d'experts informatiques résulte en la publication d'un Manifeste pour le Développement Agile de Logiciels. De ces principes naîtront des méthodes elles aussi standardisées, voire commerciales, certifiantes, qui rassureront les acheteurs informatiques. Mais est-ce que certifier des humains était vraiment le message du développement et de la gestion d'équipe itérative et Agile ?

Métriques philosophiques

De "ce qui n'est pas mesuré ne peut être amélioré" à "toute métrique sera invariablement pervertie", où est le juste milieu ? Comment reconnaître et récompenser une culture de collaboration dans des organisations où les divisions et objectifs respectifs créeent la friction ?
Agile, DDD (Domain-Driven-Design), Software Craftsmanship (artisanat logiciel), Cynefin se concentrent sur les approches humaines et reconnaissent l'art et les effets psychologiques du dialogue et du changement, de la crainte du chaos et, tout simplement, de la difficulté à gérer la complexité.
Là encore, il s'agit de découper pour tenir dans un cerveau, de comprendre et limiter les risques.

Le monde de la mode

Quand un domaine est compris et maîtrisé, on constate un sursaut de productivité là où le chaos fait place à l'ordre. On constate aussi un ralentissement et un désintérêt dans les "problèmes résolus". Comment avancer, comment progresser ?
Chaque innovation apporte un sursaut marketing court termiste, qui bien vite est soit intégré soit oublié. Il est alors temps de vendre la génération suivante sous un nouveau nom.

Exemple des ops

Par exemple, du point de vue du développeur, ASP, SaaS, Cloud ou Serverless restent essentiellement un découpage client-serveur : l'utilisateur utilise un service contrôlé par autrui, et lui développe à la fois le "back end" et un "front end" par client, en tentant si possible de profiter des différences des clients, et de partager un maximum de comportements entre chaque. Tout au plus doit-il de plus en plus parer davantage aux erreurs possibles à chacune des couches de ce mille-feuille toujours plus épais.
Du point de vue des "ops", de l'organisation et de ses achats, en revanche, chaque génération change les responsabilités : on gère son serveur chez soi, puis on confie à un prestataire la partie matérielle, infra et réseau. Puis la partie OS. Puis la partie environnement des applications. Puis les logs, les sauvegardes... le serveur autrefois chouchouté chute de son piédestal et devient un outil commun comme un autre.

Rêve du logiciel

Le logiciel n'échappera pas à cette règle : compilation, langages de haut niveau, macros, "scaffolding" (génération de parties du code par des modèles préconçus) font désormais partie du paysage, tandis que la production automatique de l'intégralité fait office de lubie inatteignable. Mais chaque génération disait cela de la suivante : le rêve de la programmation automatique ou générative, n'est pas mort, même après des étapes plus ou moins couronnées de succès, comme le MDA (Model-Driven Arcitecture), les AGL (Ateliers de Génie Logiciel), la CAO (Conception Assistée par Ordinateur ou CASE Computer-Assisted Software Engineering),
Aujourd'hui, une "usine logicielle" ou "forge logicielle" semble se limiter aux tests et déploiements continus, qui là aussi ne sont pas révolutionnaires mais existaient avant, ailleurs et sous d'autres noms.
Peut-on faire mieux ?

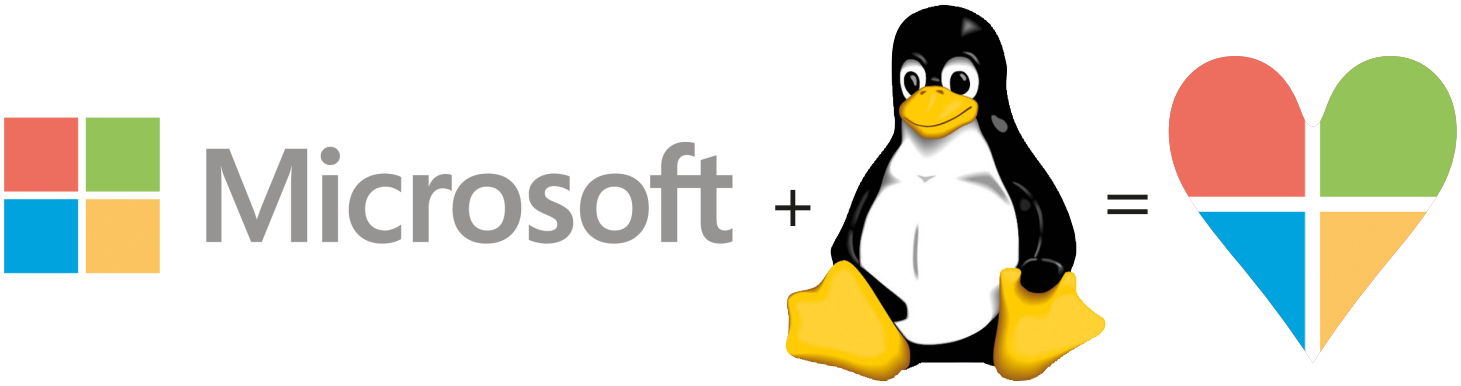
Perte de contrôle

Ces modes et standards orientent fortement vers le pré carré du voisin. Jadis représenté par IBM et Microsoft, les tenants du titre semblent actuellement plutôt être les GAFA (Google Apple, Facebook, Amazon).
Difficile de dire combien de temps une telle domination peut durer, avec des croissances toujours plus météoritiques. D'un côté, chaque génération monte plus vite et plus fort que la précédente. De l'autre, le ticket d'entrée est désormais de plus en plus cher : qui peut lutter contre des entreprises avec autant de cash et autant d'habitudes qui ramènent toujours le consommateur dans l'un des pans de leurs galaxies respectives ?

ET DEMAIN ?

Comme toujours, il sera fait d'un croisement entre l'état de l'art, les nouvelles possibilités, la société et l'économie... et le contexte de chacun ! Ce qui est innovant dans un secteur ne l'est peut-être pas dans l'autre ; ce qui est connu et admis dans votre communauté peut être une révélation au dehors.
Ni vous, ni votre voisin n'êtes en avance ou en retard : vous êtes dans un équilibre entre chacune des contraintes historiques et actuelles, en train de jongler au mieux et de prévoir afin de ne pas vous faire distancer.
Que pourront donner les croisements des mondes du développeur, du manager et de la mode, à court, moyen ou long terme ?

À bientôt pour la troisième partie ! •



En 2016, Microsoft a multiplié les annonces et les partenariats open source. L'un des moments forts fut l'arrivée comme contributeur officiel de la Fondation Linux. Que de chemin parcouru depuis la phrase choc de Ballmer en juin 2001, « Linux est un cancer » ! Le même Ballmer avait rigolé de l'annonce de l'iPhone en 2007. Dans les deux cas, il a eu tort et conduit Microsoft à des impasses. Nous nous souvenons aussi de l'agitation sur le salon « Solution Linux et Open Source » à Paris quand Microsoft avait installé son stand, les premières années.

François Tonic

Pourquoi Microsoft, qui était le symbole par excellence du logiciel propriétaire, du code fermé, de l'anti-open source à tout va, est passé de l'opposition à sa prise en compte, et, surtout, à être lui-même un contributeur actif en ouvrant de nombreux frameworks et des moteurs ? Satya Nadella, grand patron de Microsoft depuis février 2014, a été celui qui a définitivement ouvert l'éditeur sur l'Open Source. Il n'a fait que reprendre et amplifier un mouvement qui existait notamment sur la plateforme Azure. Certains diront opportunisme et business. Nous dirons pragmatisme, réalité terrain et business.

Microsoft ne pouvait plus se fermer aux nombreuses communautés open source , ni à les supporter dans les plateformes maisons comme Visual Studio ou Azure. Les frameworks et outils Open Source, notamment dans la partie serveur et infrastructure étaient incontournables. C'était une réalité que Microsoft ne pouvait plus cacher. Le pragmatisme était donc d'intégrer et de s'intégrer. Ne pas le faire c'était le risque de se marginaliser. C'est aussi aller chercher le business où il est. L'open source n'a jamais dit ne pas vendre des services, des outils, bref, faire de l'argent. Microsoft n'est pas un philanthrope. Le parallèle avec les apps et services Android et iOS est intéressant. Office sur iOS a été bloqué en partie pour ne pas nuire à la version Windows / Windows Mobile. Car Microsoft était encore centrée sur Windows. Depuis l'arrivée de Satya, Microsoft n'a fait que multiplier les apps et les services sur des plateformes non Windows, les communautés de développeurs Windows ont eu du mal à comprendre ce mouvement. Mais là encore, l'éditeur a agi avec un pragmatisme froid et logique : proposer des apps Microsoft là où les utilisateurs sont. Et dans le cas de la mobilité, ils sont sur Android et iOS.

Azure s'est « rapidement » ouvert

Une des pierres angulaires de l'ouverture Open Source a été en juin 2012 avec l'arrivée des premières machines virtuelles Linux sur (Windows) Azure ! Cette ouverture a permis de donner un socle crédible à la partie IaaS d'Azure et a ainsi permis de basculer d'une approche purement PaaS des débuts à un positionnement IaaS qui a pu positionner Azure.

Aujourd'hui, Azure supporte de nombreux outils et solutions Open Source, et pas uniquement Linux : Java, PHP, Python, Chef, Puppet, une panoplie de distributions Linux, Drupal, LAMP, MySQL, JS, NodeJS, Docker. C'est une des forces aujourd'hui d'Azure.

.Net : fermé, puis un clone ouvert, puis l'ouverture

Le framework .Net a connu une histoire mouvementée avec l'open source. Les frameworks étaient pour la plupart fermés. Quelques mois après la présentation des premières briques .Net, Microsoft avait mis en standard ouvert la partie CLI. C'est grâce à cette disponibilité que Miguel de Icaza avait initié le projet Mono dès 2001, mais il a fallu 3 ans de développement pour sortir la v1 supportant C#... Ce qui ne fut pas toujours bien accueilli par Microsoft et les relations furent parfois houleuses. Mais depuis, Microsoft a racheté Xamarin et le projet Mono est supporté par Microsoft. Les temps ont décidément bien changé…

Fondation .Net

Depuis 3 ans, Microsoft a entamé un tournant Open Source important. Non pas que l'éditeur soit uniquement Open Source, loin de là, mais il va diffuser des projets et frameworks ouverts créés par les équipes, ouvrir des pans entiers d'outils et de logiciels. Pour ce faire, Microsoft dispose aujourd'hui de deux grandes plateformes : GitHub et la fondation .Net.

La fondation .Net a été officialisée en avril 2014. Il s'agissait de créer une entité indépendante pour développer, supporter et étendre les technologies et outils autour de .Net et pour .net. Le but était de créer et de fédérer les communautés, de viser les développeurs commerciaux et open source et d'encourager la création de projets autour de .Net. L'idée derrière est de ne plus faire de .Net une entité 100 % Microsoft ni 100 % Windows.

Microsoft dépose ses propres codes ouverts,

mais la majorité des projets n'est pas d'origine Microsoft. Parmi les gros projets références : plusieurs couches Xamarin, .Net Core, ASP.Net Core, .Net Micro Framework, Roslyn.
Dernièrement, la fondation a accueilli deux éditeurs importants : Google et Samsung. Même si les deux n'évoluent pas au même niveau dans la fondation, ce sont deux arrivées non négligeables.
La fondation .Net n'a pas la prétention de rivaliser avec les grandes fondations open source tel que Linux ou Apache, mais cela peut aider .Net à sortir réellement du monde Windows, car aujourd'hui Microsoft a compris qu'il ne faut pas se focaliser uniquement sur Windows et que le monde est désormais bien plus ouvert, notamment avec les terminaux mobiles, les objets connectés. La fondation doit cependant faire ses preuves et montrer sa capacité à innover et à se démarquer de Microsoft.
Microsoft avait déjà fait une incursion sur les forges de codes ouverts avec sa propre plateforme CodePlex. Celle-ci reste disponible et utilisée par les développeurs. Mais GitHub est clairement la priorité.

On ouvre, on supporte, on contribue

Une des grosses annonces de ce changement de philosophie, l'arrivée de l'éditeur dans la fondation Linux en qualité de membre platinum. La fondation avait évoqué les nombreuses contributions sur GitHub, l'ouverture de .Net Core, le partenariat Canonical sur Ubuntu et les multiples projets Open Source. Cette arrivée est tout sauf une réelle surprise, car l'éditeur avait déjà contribué à plusieurs projets : NodeJS, OpenDaylight, Open API, Container Initiative et depuis plusieurs années sur le noyau Linux. Et même au-delà du logiciel, l'Open Hardware n'est pas oublié avec l'initiative Open Compute sur lequel Microsoft est actif sur les spécifications serveurs.
Satya Nadella n'a pas été à l'origine du mouvement ouvert, mais il l'a largement amplifié tout en se détachant d'un centrage Office – Windows pour aller totalement sur le Cloud, les services, les apps ainsi que le matériel, malgré l'échec Lumia.
Cette ouverture concerne des librairies et SDK périphériques, ne touchant pas directement les cœurs métiers et business de l'éditeur, mais aussi en ouvrant des zones entières aux outils phares : PowerShell, Visual Studio Code, le moteur JavaScript de Edge et des morceaux entiers de Xamarin, racheté en 2016.
Cette ouverture se voit dans l'intégration d'Open Source. Quelques exemples illustrent cette philosophie :

- Ubuntu Bash inclus depuis l'été 2016 dans Windows 10 ;
- Intégration des couches conteneurs Docker et aussi l'ouverture du moteur de conteneurs d'Azure (Azure Container Service) ;
- Portage de Visual Code sur des plateformes non Windows, mais cela ne suffit pas à ce que Visual Studio soit porté sur d'autres systèmes, du moins, pas à court terme ;
- SQL Server pour Linux.

L'Open Source s'est imposé à Microsoft : du *realcomputing*

Nous avions la realpolitik, nous pouvons dire que nous avons aussi du realcomputing. Le marché sans partage des années 1990 et 2000 du duo Wintel est fini. Microsoft ne règne plus en maître sur l'informatique personnelle et encore moins sur le serveur. Que ce soit dans les langages, les outils de développement, les systèmes, l'open source et les alternatives ont pris Microsoft en tenaille. L'éditeur ne pouvait plus rester fermé et refuser l'open source.
L'attaque violente de Ballmer contre Linux en 2001 était recevable à cette époque, mais a vite montré ses limites et il faut avouer que la position était intenable ! Peu à peu, l'éditeur a accepté l'arrivée de Linux et de l'open source dans les entreprises et les développeurs, et bien entendu chez les utilisateurs, notamment dans la bureautique avec des débats sans fin sur le format des fichiers. Microsoft a mis en place des licences plus permissives, ouvert des formats de fichiers et quelques morceaux de .Net. Mais le monde IT est toujours plus hétérogène et surtout les Apple, Google, Facebook, Salesforce, Amazon ont sérieusement secoué l'édifice Microsoft.
Ce changement philosophique doit être vu comme une nécessité pour répondre à la nouvelle configuration des marchés, des entreprises et des développeurs. Car l'Open Source fait partie du paysage. Mieux vaut l'intégrer et le considérer que de s'obstiner à aller contre.
Mais cela ne signifie pas que Microsoft deviendra une société open source à 100 %, loin de là. Et les frictions existent, et existeront toujours, en interne et dans certaines communautés. •

Interview de **Martin Woodward** (.Net Foundation)

Martin Woodward est executive director de la .Net Foundation chez Microsoft. Il aide les développeurs, entreprises, partenaires sur les briques .Net open source. La fondation .Net est là pour orchestrer la communauté .Net et son écosystème. Martin travaillait pour une petite entreprise que Microsoft a rachetée. Elle développait autour d'Eclipse. A son arrivée chez Microsoft, Martin a beaucoup travaillé sur les aspects Java, Git, et aidé l'éditeur à apprendre à travailler avec les communautés open source. Il fallait amener l'open source dans le business. Pas simple. Un changement philosophique important. Aujourd'hui, chaque groupe produit doit fournir des briques open source !

• Interview réalisée par Etienne Deneuve.

Etienne : ***Beaucoup de personnes s'interrogent à propos des changements chez Microsoft, par exemple l'ouverture de PowerShell, du .Net Framework. Penses-tu qu'à l'avenir Microsoft va proposer de plus en plus de contenu en Open Source ?***

Martin : En termes de contribution à l'open source, si Microsoft utilise une librairie Open Source - et Microsoft nous encourage à le faire - il nous encourage également à contribuer en retour sur ses projets pour aider par exemple à corriger certains bugs ou à faire des améliora-

tions. Pour deux raisons ; parce que c'est une bonne chose à faire, et ensuite parce que plus on contribue de façon publique, plus le projet avance et permet à Microsoft de pouvoir bénéficier plus facilement des nouveautés du projet principal. Dans le monde Open Source, ce n'est pas l'argent que tu mets dans un projet qui fait de toi un bon contributeur, mais plutôt la façon dont tu vas être impliqué dans ton projet. L'autre aspect intéressant pour Microsoft à publier du contenu Open Source, c'est surtout la capacité à diffuser de façon plus large les solutions Microsoft. Au niveau des licences nous préférons utiliser la licence MIT, car elle nous permet de ne pas accepter toutes les modifications dans les contributions.
Ceci parce que, par exemple, le SDK d'Azure doit fonctionner puisque des personnes payent pour ça ; cela ne devant pas l'empêcher néanmoins de rester compatible avec les autres projets Open Source. Le dernier point c'est là aussi la collaboration. Par exemple Type Script ou Chakra, qui sont des technologies développées chez Microsoft, sont différentes de la plateforme .Net où il y a toujours un écart significatif entre ce qui sort de chez nous et ce qui est en production réelle chez nos clients. Avant, .Net était livré dans chaque version de Windows, et ce n'était pas un problème. Mais maintenant le "Time-To-Market" est devenu beaucoup plus important et plus court ; les mises à jour du .Net sont désormais disponibles.
Il y a eu des gros changements depuis la communauté dans le .Net Core par exemple qui ont permis de résoudre des problèmes dans la mise à l'échelle ou des optimisations dans les performances. Dans Type Script c'est pareil, des bonnes choses viennent de la communauté. Rendre le .Net open source, ce n'était pas pour gagner plus d'argent, mais pour rendre disponible plus largement et profiter des retours et de nouvelles idées.

E : ***Est-il difficile pour toi, de faire ton travail dans une entreprise qui était le diable il y a quelques années pour les communautés de l'open source ?***
M : Pour moi, l'ouverture a été une joie. Avant ça, j'expliquais à mes collègues comment fonctionnaient les communautés autour de Git, et souvent j'étais un peu seul dans les réunions (rires), c'était marrant. Souvent, les équipes oublient qu'il faut communiquer avec non seulement le code mais aussi les road maps. Il reste beaucoup de changements à faire. Il nous reste beaucoup à apprendre sur le comment de la collaboration dans le style de l'open source. C'est fascinant de voir les personnes évoluer vers ce modèle dans une grande entreprise. Maintenant par défaut tous les codes sources sont disponibles pour tout le monde en interne, je peux par exemple voir tout le code source de Windows et collaborer.
Un autre exemple marrant, a été le changement au niveau des mails pour avertir les collègues d'une naissance. Au début les personnes répondaient "joli bébé... Bla Bla" et un jour ce type mail est arrivé dans l'équipe .Net, les réponses sont devenues "+1 Looks good to Merge".(rires)

E : ***Qui peut contribuer sur les projets Open Source de Microsoft ? Tout le monde peut participer ?***
M : C'est aussi quelque chose qu'on a changé. Avant il fallait beaucoup d'autorisations pour changer ne serais-ce que quelques bytes. Je l'ai vécu à l'époque où je travaillais avec Eclipse pour le faire fonctionner mieux sur des Windows 64bits. Aujourd'hui il suffit de dire au Manager, tiens j'ai trouvé cette amélioration tu en penses quoi ? "c'est cool, vas-y" C'est beaucoup plus facile aujourd'hui qu'il y a quelques années, on a mis en place des processus simplifiés... Y compris pour publier des éléments, il suffit de demander au manager si cette partie peut être open source ou pas.

E : ***On se demande souvent pourquoi Microsoft publie des applications sur Android, iOS, macOS ... Quel intérêt ?***
M : C'est souvent pour le business. En général ça fait partie d'une souscription, donc si ce n'est pas disponible sur les autres plateformes, ça risque d'être moins facile de placer par exemple Office 365 si les utilisateurs Android ne peuvent pas accéder à leurs données. L'idée est d'avoir le plus de personnes possibles sur la plateforme. Microsoft c'est aussi une entreprise d'outils de productivité avec par exemple des outils de collaboration. Il serait donc incohérent et impossible de réussir cette mission, si on n'était pas cross plateforme. C'est depuis longtemps une réalité ; par exemple le client Lync sur MacOs fonctionne très bien.

E : ***Pourquoi l'équipe qui est derrière VS Code a utilisé un Framework GitHub au lieu de prendre un Framework traditionnel de Microsoft ?***
M : En fait, j'étais dans le projet. L'idée était de proposer un outil pour développer rapidement des petits sites Internet. Le Framework Electron était déjà là, il fonctionnait bien. On a donc décidé de mutualiser les efforts pour le rendre meilleur sur certains aspects comme l'accessibilité. Electron est présent dans beaucoup de solutions comme Slack, qui l'utilise aussi.

E : ***Comment Microsoft collabore-t-il avec les projet Open Source, juste en contribuant ?***
M : On utilise par défaut des licences MIT, mais on participe sur tous types de licences. Linux c'est du GPL, Git c'est un mélange de GPL et de LGPL. On ne pose pas simplement notre code dans un GitHub public mais on travaille dedans réellement. C'est pas un dump qu'on pose là et qui reste statique. On open source la totalité, y compris la documentation etc. etc., ça demande du temps.

E : ***Coté développeur, il y a beaucoup plus de chose que pour le coté IT, est ce que ça va changer ? Quelle est la position ?***
M : Pour l'instant PowerShell a été rendu Open Source, c'est déjà un bon début ; il tourne sur toute les plateformes... Qu'entends-tu par là ?

E : ***Par exemple, Docker comment ça se passe pour intégrer une technologie comme ça dans Windows ?***
M : Une première façon de le faire c'est de donner du temps d'ingénierie. Sinon on peut aussi joindre une fondation par exemple comme la Linux Fondation. C'est intéressant pour les communautés d'avoir des personnes qui peuvent passer du temps sur ces projets. C'est difficile dans les milieux open source d'avoir des ressources à temps plein.

E : ***Tu penses que Windows va devenir Open Source à l'avenir ?***
M : Il y a déjà des parties de Windows qui le sont, cela dit, je ne sais pas tout ce qui va se passer pour cette partie, surement s'il y un intérêt business pour ça.

E : ***Pour conclure, dirais-tu que Microsoft n'est plus le diable finalement ?***
M : Microsoft est toujours dans le business pour faire de l'argent. S'il y a des participations aux projets open source c'est souvent pour le business. Mais si c'est fait de façon responsable, et que la communauté s'enrichit, il n'y pas de problème. Basons-nous donc sur les faits et pas sur l'histoire… •

Open Source & Azure

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

Même si vous n'êtes pas d'accord, l'Open Source est le grand gagnant de ce siècle, car l'Internet va vite, même très vite. Il est difficile de concevoir un nouveau logiciel ou une application si vous souhaitez communiquer à travers Internet, et au lieu de réinventer la roue, il est préférable d'utiliser les solutions ouvertes et donc open source.
Grâce à l'open source, vous avez à disposition de nombreuses ressources aussi bien sur le plan humain que documentaire, que sur le plan des tutoriels pour apprendre, utiliser et déployer des applications sur n'importe quels environnements : ordinateurs de bureau, portables, tablettes, smartphones...

Changement de comportement

L'Internet d'aujourd'hui a fait abstraction de la notion d'OS (système d'exploitation) que nous connaissons (Windows, Linux, Mac OS) pour laisser la première place aux navigateurs. C'est pourquoi, de nombreux logiciels offrent des versions Web comme Office 365 et ne se limitent plus aux ordinateurs individuels.

Une architecture

Pour répondre au mieux aux nouvelles utilisations, l'architecture a un rôle primordial et doit s'adapter pour proposer une solution solide, car c'est un élément clef lors de la réalisation d'un projet métier. Il existe différentes solutions modulaires pour permettre à tous les développeurs de configurer facilement cette architecture comme le propose Microsoft Azure. Celle-ci permettra d'augmenter et de réduire les ressources (processeurs, mémoires) à travers une interface sans avoir à compiler et suivre des procédures de validation longues et fastidieuses. Ces avantages anticipent les hausses et les baisses d'utilisation du projet.
Bien entendu, d'autres critères doivent être pris en compte comme la possibilité d'ajouter un ou plusieurs serveurs à tout moment dans un cluster suivant l'importance du projet.

La réalisation

La réalisation d'un projet Web doit être pensée aussi bien du côté back-office, que du côté front-office avec la possibilité d'utiliser les solutions Open Source en ligne de commande (terminal) ou par une interface en ligne. Celui-ci peut difficilement se limiter à un seul langage ou une seule technologie. Mais pour qu'une application ou projet Web fonctionne, il est indispensable de posséder un système d'exploitation qui corresponde à vos attentes ; vous le trouverez sous Linux ou en gardant Windows azure. Pour réaliser ce projet Web, vous associez différentes briques logicielles Open Source pour obtenir par exemple un serveur, une base de données et un langage pour avoir un environnement AMP (Apache, MariaDB, PHP). De plus, Il est possible d'ajouter un logiciel de versioning (GIT, SVN), ou encore des options de caches pour améliorer les performances et les temps de réponses. Bien entendu, la base de données peut avoir son propre cluster, découpée en plusieurs serveurs Maîtres / Esclaves, le tout géré en load balancing pour communiquer avec le langage de développement. Pour éviter de recréer les fonctionnalités, vous utilisez une ou plusieurs applications suivant votre besoin, comme un gestionnaire de contenu (CMS), ou générique (framework). Mais vous appelez aussi des logiciels externes pour améliorer les fonctionnalités de celui-ci comme pour effectuer des recherches dites « complexes » (facettes, avancées…) ou encore pour bénéficier de l'utilisation de machine learning ou de ressources de documentaires.
Par ailleurs, pour mettre à disposition les pages Web de votre projet, vous pouvez déployer avec des scripts ou des scénarii automatiques, vers des serveurs frontaux pour que les internautes puissent l'utiliser. Pour cela, vous utiliserez des logiciels de déploiement (Ansible, Puppet, Chef…). Du côté du front-office, celui-ci sera appelé par un navigateur ou une application (APPs) mobile. Pour cela, l'architecture doit être légère, avec un temps de réponse réduit. Il doit aussi offrir une navigation fluide et minimaliste. C'est pourquoi les nouveaux langages HTML5, CSS3 et JS seront utilisés.

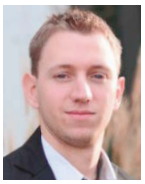
Au final

Pour communiquer avec toutes ces solutions ouvertes aussi bien pour contrôler le matériel, que le code, Azure s'est rendu compatible avec l'ensemble de ces normes Web. Grâce à cela, de nombreux moyens de communications sont disponibles pour échanger les données entre les environnements et les applications.
La plateforme applicative Microsoft Azure saura répondre à ces différentes solutions suivant votre besoin comme au niveau des appels aux Web Services (Rest), HTTP (jSON), aux applicatifs (métier, services, données), au besoin en cache (Services Workers).

Enfin

Pour ma part, l'open source n'est pas une nouvelle solution car elle a toujours existé depuis que l'informatique grand public est apparue. Mais elle a trouvé sa place légitime avec la naissance de l'Internet pour pouvoir contribuer, partager et tester des nouvelles solutions propriétaires et libres. Par ailleurs, avec l'ouverture de la plateforme Azure, celle-ci est aussi une solution d'hébergement et de travail distant. •

.Net Standard 2.0 et son utilisation dans les projets .NET



Christophe Gigax

Co-Fondateur du Microsoft User Group de Strasbourg
Auteur du livre sur ASP.NET Core MVC : Maîtrisez ce Framework Web puissant, ouvert et multiplateforme

La portabilité des codes sources et des librairies a toujours été une problématique récurrente au fil des années. Comment rendre mon code réutilisable entre plusieurs projets ? Entre plusieurs plateformes ? Est-ce que la maintenance de ma librairie sera la même entre les différentes plateformes cibles ? L'écosystème de Microsoft grandit rapidement, et cette problématique prend de plus en plus de place au sein d'une équipe de développement. Comment partager efficacement mon code C# entre un projet Xamarin et un projet ASP.NET Core ? Annoncé en Septembre 2016, .Net Standard est la solution à ce problème : le partage du code de manière universelle, peu importe la plateforme ciblée.

Etat des lieux

Depuis les débuts de .NET, les développeurs sont capables de créer des bibliothèques de classe dans leurs solutions, et ainsi partager le code C# entre différents projets. Ces projets particuliers, appelés « **Class Library** », sont avantageux car ils réduisent les temps de développement du code métier et permettent la réutilisabilité des classes déjà existantes. Ce type de projet possède tout de même un désavantage : il ne permet de cibler qu'une seule plateforme à la fois.

Les plateformes Microsoft sont découpées en 3 grandes familles : .NET Framework, .NET Core et Xamarin. Chacune possède ses propres bibliothèques de base permettant au développeur de concevoir une application UWP ou un site Web en ASP.NET. Cependant, il n'est pas possible d'avoir une bibliothèque de classe référencée en même temps par une application UWP et un site Web ASP.NET, car les bibliothèques de base sous-jacentes ne sont pas les mêmes. Avec l'apparition de .NET Core et le rachat de Xamarin par Microsoft, les plateformes sont cloisonnées, possédant chacune leurs propres caractéristiques. [1]

Plusieurs solutions existent aujourd'hui afin de pallier ce problème. La première s'intitule la Portable Class Library (PCL) et permet de cibler plusieurs plateformes dans une même bibliothèque de classes. Les plateformes ciblées se définissent dans les paramètres du projet.

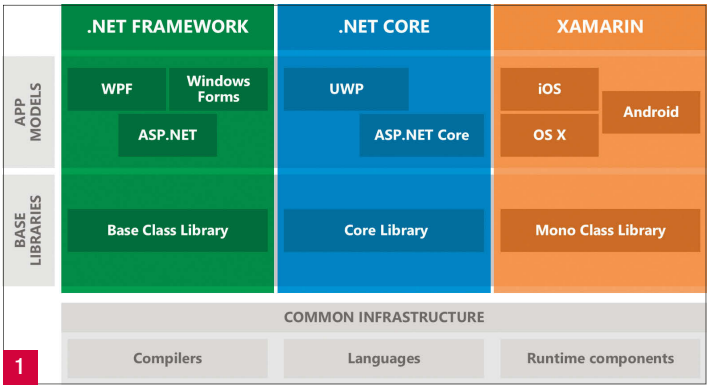
La règle avec les PCL est simple : plus il y a de plateformes ciblées, plus le champ d'action de la bibliothèque sera restreint. En effet, si la PCL cible Xamarin.Android et Silverlight 5, la PCL ne pourra pas utiliser certaines classes de MonoDroid car elles ne sont pas disponibles sur Silverlight 5. Le grand avantage des PCL est son côté universel entre diverses plateformes et sa testabilité. La deuxième solution concerne les projets partagés, appelés les « **Shared project** ». Ces derniers sont un type de projet particulier permettant de partager du code sans restriction : les fichiers sont simplement copiés dans les projets référençant le projet partagé. Le développeur doit alors utiliser des directives de compilation afin d'adapter son code aux différentes plateformes. Toutes ces solutions ne sont pas réellement adaptées au partage d'un code unique et multiplateforme sans concession. Le but étant d'avoir un code hautement maintenable et très vite disponible entre les différentes plateformes, les restrictions de la PCL et les directives des projets partagés ne sont pas la solution miracle au partage de code. C'est pourquoi Microsoft propose depuis peu un nouveau standard dans l'écosystème .NET : .Net Standard.

.Net Standard, c'est quoi ?

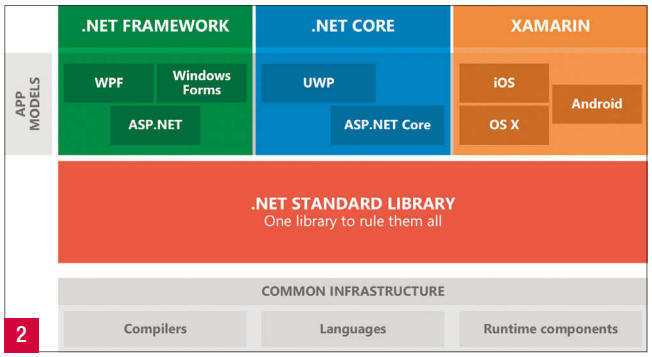
.Net Standard est un nouveau jeu d'APIs que toutes les plateformes .NET vont devoir implémenter. Les plateformes ciblées sont : .NET Core, .NET Framework et Xamarin, et vont ainsi permettre d'unifier les plateformes autour d'une librairie .NET unique. A terme, .Net Standard devrait remplacer les PCLs. Les développeurs peuvent suivre l'avancement de .Net Standard sur GitHub : https://github.com/dotnet/standard. [2]

L'intérêt de .Net Standard est le partage de code sans concession, c'est-à-dire que les APIs disponibles seront utilisables sur n'importe quelle plateforme .NET sans aucune restriction. Pour les entreprises, il est alors possible de construire un socle de code solide et réutilisable rapidement sur une autre plateforme de type desktop, Web ou mobile.

La première version de .Net Standard est sortie en même temps que .NET Core, c'est-à-dire en Juin 2016. Depuis, .Net Standard a bien évolué



Cloisonnement des plateformes



Modèle de programmation avec Net Standard

et nous en sommes aujourd'hui à la version 1.6, rajoutant ainsi plusieurs sets d'APIs au fil des versions. Cependant, toutes les plateformes ne supportent pas encore ces nouvelles APIs. Voici un tableau récapitulatif des versions supportées par les plateformes :

.NET Platform	.NET Standard							
	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	→	→	→	→	→	→	1.0	vNext
.NET Framework	→	4.5	4.5.1	4.6	4.6.1	4.6.2	vNext	4.6.1
Xamarin.iOS	→	→	→	→	→	→	→	vNext
Xamarin.Android	→	→	→	→	→	→	→	vNext
Universal Windows Platform	→	→	→	→	10.0	→	→	vNext
Windows	→	8.0	8.1					
Windows Phone	→	→	8.1					
Windows Phone Silverlight	8.0							

Les flèches indiquent que la plateforme supporte une version plus récente que celle indiquée dans la colonne. Par exemple, .NET Core supporte toutes les versions de .Net Standard, alors que le .NET Framework 4.5 ne supporte que la version 1.1 de .Net Standard.

Avec ce tableau, le développeur peut ainsi savoir quelle version de .Net Standard il peut cibler. S'il utilise un projet sous .NET Framework, il doit bien faire attention à son ciblage sous peine d'avoir des incompatibilités au niveau des APIs. Concernant Xamarin, la seule version de .Net Standard compatible sera la version 2.0, qui devrait sortir au premier ou au deuxième trimestre de 2017.

Les nouveautés de la version 2.0

Les projets avec les versions actuelles de .Net Standard sont capables de référencer d'autres librairies du type :

- **.Net Standard**, si la version est inférieure ou égale à celle de la version actuelle de la librairie ;
- **PCL**, si cette dernière cible .Net Standard avec une version inférieur ou égale à la version de la librairie.

Cependant, cette comptabilité ne reflète en rien la réalité du marché du NuGet. En effet, la plupart des librairies disponibles sur NuGet ciblent le .NET Framework, et la plupart des APIs de .Net Standard ne sont pas encore complètement intégrées au nouveau standard de Microsoft. Le tableau ci-dessous récapitule le taux d'adoption des PCL et de .Net Standard par rapport au .NET Framework :

Target	Occurrences
.NET Framework	46,894
.NET Standard	1,886
Portable	4,501

De ce fait, .Net Standard 2.0 intègrera la possibilité de cibler nativement des librairies .NET Framework, étendant les possibilités de manière fulgurante au vu du nombre de librairies ciblant cette plateforme aujourd'hui sur NuGet. Evidemment, cette comptabilité sera possible uniquement si la librairie .NET Framework expose des APIs utilisables par la librairie Net Standard. Cette nouveauté est la première de .Net Standard 2.0.

Ensuite, via le tableau croisé des plateformes .NET et de .Net Standard présenté dans la section précédente, on peut voir que .Net Standard ne sera supporté que par la version 4.6.1 du .NET Framework.

Ce choix de Microsoft repose sur un constat simple : c'est la version du Framework la plus utilisée actuellement, et cibler une nouvelle version du Framework ne fera pas adopter plus largement .Net Standard auprès des développeurs. Pour ce faire, les équipes ont dû supprimer les APIs de .Net Standard introduites durant les versions 1.5 et 1.6. Ceci est un **breaking changes**, cependant les impacts sont minimes puisque seulement 6 packages sur NuGet ciblent ces versions. Les équipes ont séparé les APIs disponibles via .Net Standard en 2 parties :

- **Les APIs obligatoires** : elles seront délivrées automatiquement avec Net Standard, et ce sont celles qui peuvent être implémentées de manière cross-platform à travers Xamarin, .NET Core et .NET Framework ;
- **Les APIs optionnelles** : elles seront spécifiques à la plateforme et seront délivrées via NuGet dans des paquets séparés.

Ainsi, plusieurs APIs centrales du Framework ont dû être revues afin de respecter cette hiérarchie, comme *Code Access Security (CAS)* qui est devenue optionnelle et doit ainsi être séparée de *App Domain* qui, elle, reste dans Net Standard. Le schéma ci-dessous récapitule sommairement les APIs disponibles via .Net Standard 2.0 : [4]

Le concept autour de .Net Standard est donc très simple. La version 2.0 va fournir des APIs très généralistes et utilisables sur toutes les plateformes, afin de mutualiser un maximum de code dans ces librairies. Ensuite, plusieurs paquets NuGet seront des extensions de Net Standard, spécifique à chaque plateforme afin de rajouter des APIs. Enfin pour finir, certaines APIs seront incluses dans .Net Standard mais risquent tout de même de ne pas être disponibles sur toutes les plateformes. En ces rares occasions, l'API pourrait lever une exception de plateforme.

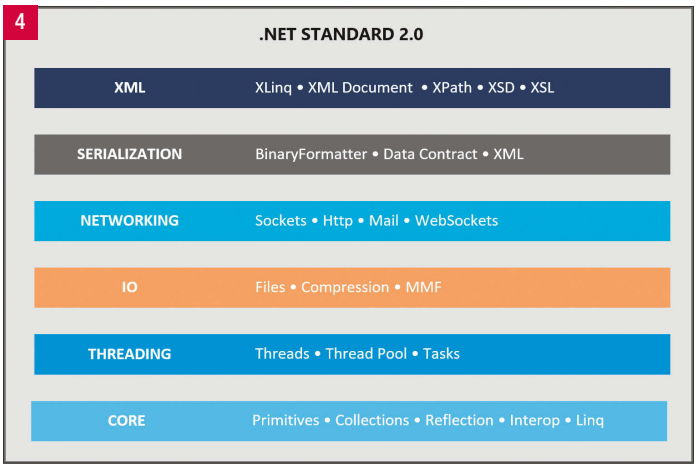
Comment bien utiliser .Net Standard 2.0

La création d'une librairie de projet .Net Standard ne sera disponible qu'avec Visual Studio 2017, anciennement intitulé VS '15'. La nouvelle version est censée remplacer les PCLs, et donc le développeur doit prendre en compte ce changement dans ses projets .NET. Microsoft fournit un utilitaire afin de vérifier la comptabilité de votre code avec .Net Standard : https://github.com/Microsoft/dotnet-apiport. Via la commande suivante, le développeur est capable de vérifier s'il doit attendre la version 2.0 avant de migrer sa librairie ou non, en vérifiant aussi la compatibilité de sa librairie avec la version 1.6 :

```
> apiport analyze -f C:\src\mylibs\ -t ".NET Standard,Version=1.6"^
                  -t ".NET Standard,Version=2.0"
```

Concernant la migration depuis une PCL, Visual Studio propose une migration directement depuis les propriétés du projet PCL. Faites **Clic-droit sur le projet** > **Propriétés**, puis cliquez sur **Target .NET Platform Standard**. Visual Studio va ainsi automatiquement transformer votre librairie PCL en librairie Net Standard. •

4



APIs disponible via Net Standard

Vcpkg, un outil pour acquérir et compiler plus simplement les librairies open source C++ sur Windows.

Eric Mittelette
Microsoft Corp.

Acquérir une librairie open source, la compiler sur Windows et l'intégrer dans son projet C++ reste une opération délicate, voire une galère. Pour une librairie donnée, il faut trouver les sources, les installer localement, compiler la librairie et enfin la mettre à disposition du projet dans lequel vous souhaitez l'utiliser.

La phase de build est de loin la plus subtile et complexe, elle nécessite souvent un peu d'adaptation (patch) si la librairie n'est pas encore disponible pour la version de compilateur que vous utilisez, cette adaptation nécessite encore l'usage d'incantations (sous forme de scripts, pas de magie noire…), que seuls les grands « faiseurs » maîtrisent réellement. Nous savons par sondage et les appels au support technique de Microsoft que les librairies tierces restent pour plus de 30% des cas le bloqueur a la migration vers les dernières versions du compilateur C++. Nous savons également que 80% des projets C++ utilisent 2 ou 3 librairies tierces en moyenne, et que la vaste majorité d'entre elles sont aujourd'hui des librairies open source. Intéressant de noter que cette proportion s'est inversée ces 10-15 dernières années ;, au départ c'étaient essentiellement des librairies propriétaires qui étaient utilisées par les développeurs C++. Aujourd'hui la seule solution proposée par Microsoft pour acquérir des librairies tierces C++, est NuGet, mais NuGet n'a pas été pensé pour le code natif, mais pour le code managé. Bien sûr il est tout à fait possible d'utiliser NuGet, mais il est important de noter que le package a été produit pour une ligne de compilation donnée (si vous utilisez d'autres options pour votre code, ça ne marchera pas correctement !) et comme vous n'avez pas accès au code source, vous ne pourrez pas utiliser cette librairie. Or nous savons grâce aux enquêtes et interviews réalisées que les développeurs C++ veulent avoir accès aux codes sources, soit pour debugger, soit tout simplement pour vérifier la qualité du code qu'ils s'apprêtent à injecter dans leur projet. Fort de ces constats, nous avons décidé de proposer une solution pour Windows. Le parti pris pour le design était de proposer une solution open source ou tout le monde peut participer, et de s'inspirer de ce qui marche sur les autres plateformes telle Linux ou Mac. Nous avons passé un bon moment à regarder Debian, app-get ou Homebrew, pour vérifier nos hypothèses, issues de nos dernières enquêtes sur le sujet :

- 80% des projets VC++ utilisent 2 à 3 librairies externes, donc plus de 80% sont des librairies en open source ;
- Les Dev C++ veulent avoir accès au code, pour des raisons de revues de qualité ou de débogage ;
- Maintenir un ensemble de librairies stables entre elles est un point critique (« library hell ») ;
- Proposer un ensemble de librairies sur les versions les plus récentes ;
- Il est important de pouvoir disposer de plusieurs versions d'une même lib sur un même poste (pas dans un même exécutable) ;
- Les développeurs doivent pouvoir modifier à loisir les options de compilation ;
- Les librairies doivent être utilisable en statique ou dynamique (Link).

Architecture globale

Vcpkg a été conçue sur une approche « port tree ». Wikipedia (https://en.wikipedia.org/wiki/Ports_collection) définit cette approche comme suit : « un ensemble de makefile et patches …. comme méthode simple pour installer des logiciels ou créer des packages binaires ». Un « port file » décrit comment compiler une libraire, il ne contient pas le code source de la librairie. Nous voulions que cette collection de « port files » soit open source et que la communauté des développeurs C++ puisse y contribuer en ajoutant les librairies qu'ils maintiennent ou utilisent.

L'architecture générale en découle assez naturellement :

- Créer une liste de « port file » dans un repo Github ;
- Créer un outil en ligne de commande qui exploite cette liste et capable de lancer le processus de compilation au regard des instructions de chaque « port file » ;
- Installer le résultat de la compilation dans un répertoire local « « LibFolder » ;
- Permettre une intégration simple avec Visual studio, CMake ou tout autre build system de ce « LibFolder ».

Le « LibFolder » est distribuable par simple Xcopy, ou partageable sur un réseau (pas de registry, gac…). Le « LibFolder » est indépendant de Vcpkg ; on peut donc ajouter ce que l'on veut dedans, comme par exemple des librairies propriétaires ou personnelles. Le « LibFolder » peut être vu comme un cache global pour les librairies C++. Un « LibFolder » est donc un ensemble de librairies stables et compatibles les unes avec les autres, il n'y aura donc qu'une seule version d'une librairie donnée dans un « LibFolder ». Si l'on souhaite une seconde version (ou plus) d'une même librairie, il faudra créer un nouveau « LibFolder ». Vcpkg ne propose pas :

- Un format d'archive. Le « LibFolder » n'est pas compressé ;
- Un packaging des binaires. Même s'il est tout à fait possible de placer dans le « LibFolder » ce que l'on veut, comme des librairies propriétaires en binaire (exposant .h et .lib et dll).

Vcpkg en action

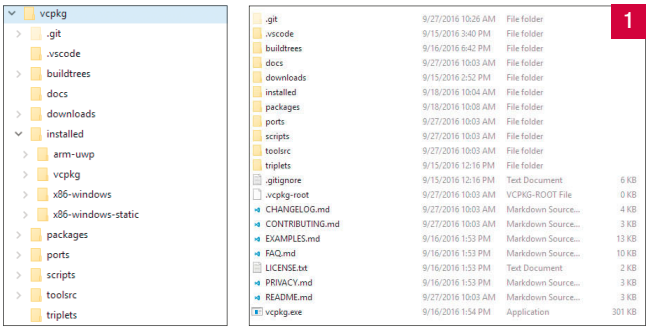
La première chose à faire, pour utiliser Vcpkg, est d'installer l'outil en clonant le repo GitHub :

```
git clone https://github.com/microsoft/vcpkg.git
```

Ensuite il faut compiler Vcpkg lui-même en exécutant cette commande depuis le répertoire ou a été cloné le repo :

```
powershell -exec bypass scripts\bootstrap.ps1
```

Vcpkg.exe va alors être généré ainsi que le « LibFolder », celui-ci aura la hiérarchie suivante : [1]. Les librairies installées sont dans le répertoire **installed**, soit en release, soit en debug. Pour une cible donnée (dans la



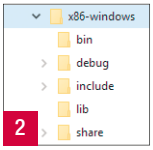


figure ci-dessous x86-windows) vous trouverez le répertoire **include** contenant les headers(.h), le répertoire **lib** contenant les (.lib), et le répertoire **bin** contenant les dll, idem pour le mode debug. [2]

Installer et utiliser une librairie avec Vcpkg

Rien de plus simple ! A quand remonte votre dernière expérience en la matière ? Avez-vous déjà essayé d'installer Boost ou OpenCV ? Si c'est le cas vous devez vous rappeler que ce n'est pas toujours trivial ! Avec Vcpkg il suffit de faire appel à la commande **install** :

```
Vcpkg install boost
```

Et voilà c'est fait ! Bon vous aurez le temps de boire un café, voir la cafetière selon la puissance de votre machine, car il faut du temps pour compiler OpenCV ou Boost ;-) Mais cette commande suffira ! Vcpkg travaille de manière récursive et installera en premier les librairies dont dépend la librairie que vous souhaitez installer. En fait la première commande que vous allez utiliser sera :

```
Vcpkg search
```

Celle-ci vous permettra de lister toute les librairies disponibles (à la date de rédaction de cet article elles sont au nombre d'environ 100). Le nom listé est celui à utiliser pour installer la librairie, la commande liste également le numéro de version et une description sommaire de quelques mots. Par défaut c'est la version x86 dynamique (dll) qui est installée, pour préciser la plateforme cible on ajoutera un triplet :

```
Vcpkg instal zlib :x64-windows
```

La liste des triplets disponibles est dans le répertoire **triplets :** x64-uwp.cmake ; x64-windows-static.cmake ; x64-windows.cmake ; x86-uwp.cmake ; x86-windows-static.cmake ; x86-windows.cmake.
Note, il est possible d'installer plusieurs librairies en même temps :

```
Vcpkg install boost, openCV, SDL
```

On pourra dans cette même commande préciser les architectures cibles. Une fois installée, il faut rendre les librairies disponibles pour Visual Studio 2015 ou VS2017 car seules ces versions sont supportées et utilisées. Visual Studio 2015 et 2017 utilisent la même version de la CRuntime, n'avoir qu'une seule version de la CRuntime pour toute les librairies est une garantie d'inter compatibilité.
Pour intégrer la LibFolder avec Visual Studio deux possibilités :

- Au niveau utilisateur, la commande :

```
Vcpkkg integrate install
```

  Elle va rendre l'ensemble des librairies de la LibFolder disponible pour tous projets. Inutile de refaire appel à cette commande lors de l'ajout de nouvelles librairies, l'intégration se fait une fois pour toute (et nécessite des droits administrateur la première fois qu'elle est exécutée).
- Pour obtenir une intégration projet par projet, ce qui permettra d'avoir plusieurs LibFolder sur une même machine, pour par exemple avoir plusieurs versions différentes d'une même librairie, on utilisera alors la commande suivante :

```
Vcpkg integrate project
```

Cette commande va créer un package Nuget contenant en fait une simple page de propriété (property sheet). On ajoute cette page de propriété via le NuGet package manager de Visual studio.
La commande **integrate** propose une option **remove** :

```
Vcpkg integrate remove
```

Cette commande va supprimer uniquement l'intégration au niveau utilisateur, pour les intégrations au niveau projet il faudra désinstaller le package NuGet du projet. Il ne peut y avoir pour un projet donné deux intégrations, donc si vous ajoutez un package NuGet, seul ce dernier compte et l'intégration niveau utilisateur est supprimée du projet courant. On peut tout à fait utiliser Vcpkg et son LibFolder depuis CMake, il suffit alors d'ajouter le fichier toolchain (`scripts\buildsystems\vcpkg.cmake`) dans votre fichier CMake :

```
-DCMAKE_TOOLCHAIN_FILE=D:\src\vcpkg\scripts\buildsystems\vcpkg.cmake
```

Le fichier examples.md sur github [https://github.com/Microsoft/vcpkg/blob/master/docs/EXAMPLES.md] explique plus avant la mise en œuvre.
La commande update, ne va pas modifier le contenu du LibFolder (ce serait trop dangereux, aucune mise à jour est automatique par design). Cette commande va juste lister les librairies qui ont été mises à jour sur le repo GitHub. C'est au développeur de faire ou non la mise à jour. [3]

Ajouter une nouvelle librairie : création d'un nouveau port file

Pour ajouter une librairie au catalogue il faut fournir :

- Un fichier de contrôle :
  - Très simple il contient le nom, la version, une courte description et les éventuelles dépendances de la librairie.
- Un port file :
  - Ecrit dans la syntaxe de CMake il permet toutefois de faire appel tous type de build system. Il faut :
    - Préciser ou télécharger les sources ;
    - Indiquer comment les décompresser et les préparer pour la compilation ;
    - Appliquer les patchs le cas échéant ;
    - Lancer le processus de build.
- Des fichier patchs :
  - Contiennent les instructions à appliquer pour préparer le projet à être compile avec VC2015.

Un exemple de création de port file et de patch file sont disponibles sur GitHub.

Contribuez !

Les migrations vers les dernières versions du compilateur C++ sont souvent bloquées par des librairies tierce parties, leur mise à jour est souvent fastidieuse, avec Vcpkg nous avons essayé de simplifier cette acquisition et d'optimiser vos migrations. Le projet étant open source, la richesse du catalogue est donc directement liée à la participation de la communauté C++. Contribuez en ajoutant les « port file » correspondant à vos librairies, ou celles que vous maintenez. Vous pouvez aussi créer une issue pour demander une nouvelle librairie ! C'est collectivement que nous fabriquons et enrichissons le catalogue des librairies open source disponibles sur Windows. Comme pour tout projet open source, faites-nous part de vos remarques, suggestions ou bug en créant des issues ou des pull request.
A bientôt sur https://github.com/Microsoft/vcpkg/ •

```
C:\temp\vcpkg>vcpkg update
Using local portfile versions. To update the local portfiles, use `git pull`.
The following packages differ from their port versions:
    boost:x86-windows           1.62-2 -> 1.62-4
    cpprestsdk:x86-windows      2.8 -> 2.9.0-1
    glog:x86-windows            0472b91 -> 0.3.4-0472b91
    sdl2:x86-windows            2.0.4 -> 2.0.5

To update these packages, run
    vcpkg remove --purge <pkgs>...
    vcpkg install <pkgs>...
```
3

MODÉLISATION DES DONNÉES

Partie 4

Quels intérêts et usages pour le **reverse engineering** ?

Steve Berberat
Collaborateur scientifique
Haute école de gestion Arc, HES-SO
steve.berberat@he-arc.ch

Pierre-André Sunier
Professeur HES
Haute école de gestion Arc, HES-SO
pierre-andre.sunier@he-arc.ch

En qualité de développeur, il nous est parfois demandé de reprendre une application faite sur mesure. Pour la comprendre, nous devons lire son modèle de données. Hors, il se peut que ce dernier ne soit pas disponible ou soit désuet. La solution : recourir à l'ingénierie inverse.

Base de données d'exemple

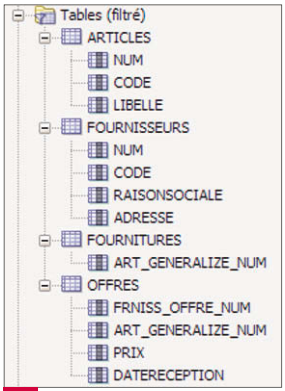
Pour illustrer le fonctionnement de l'ingénierie inverse, nous nous baserons sur une petite base de données Oracle existante et dotée de 4 tables. La Figure 3 présente la visualisation des tables et des colonnes de cette base de données à partir de SQL Developer. Volontairement, aucun modèle de données n'existe pour le moment : ce sont justement les fonctionnalités d'ingénierie inverse qui nous permettront d'en créer et de mieux comprendre cette structure de données !

Illustration avec l'automate MVC-CD

Nous allons réaliser l'ingénierie inverse en générant les modèles logiques et physiques, puis conceptuels de données, avec deux outils différents. Dans ce chapitre, nous présentons cette démarche et le résultat obtenu avec un premier outil appelé « MVC-CD ».

L'automate de transformation MVC-CD

L'automate « MVC-CD » est un prototype développé dans le cadre d'un projet de recherche à la Haute école de gestion Arc. Il se matérialise sous la forme d'un plug-in à ajouter à Visual Paradigm, qui est un outil de modélisation supportant plusieurs types de diagrammes dont notamment ceux proposés par UML. Le plug-in est téléchargeable librement. Vous trouverez toutes les informations nécessaires dans les références de cet article. Veuillez noter qu'un des atouts de cet automate est de pouvoir générer une base de données relationnelle Oracle embarquant les triggers et procédures qui assurent le respect des contraintes et règles métiers décrites initialement sur le MCD. Si cet aspect vous intéresse, sachez qu'il a fait l'objet d'un de nos précédents articles publiés dans l'édition de novembre 2016.



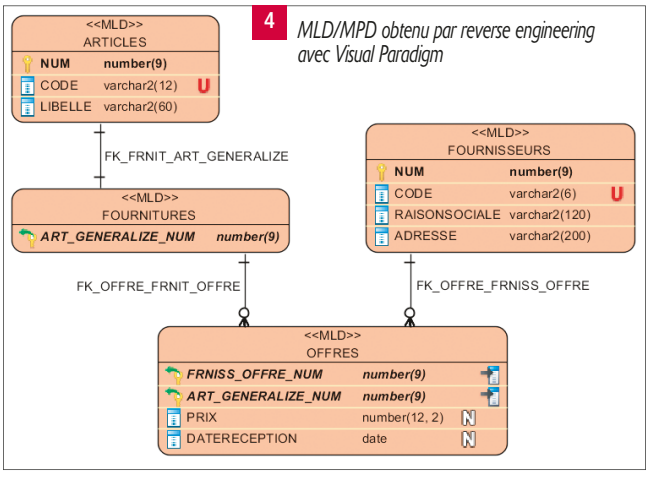
3 *Base de données d'exemple*

Générer le MLD et le MPD

Visual Paradigm effectue nativement de l'ingénierie inverse à partir d'une base de données vers des modèles logiques et physiques ; le plug-in n'est donc pas encore nécessaire à ce stade. La fonctionnalité est disponible dans le menu « Tools » puis « DB ».
Après avoir renseigné les informations de connexion à la base de données, nous devons sélectionner le schéma Oracle dans lequel Visual Paradigm ira rechercher la structure de données. La liste des objets présents dans ce schéma est affichée et nous devons cocher ceux que nous souhaitons prendre en compte pour réaliser l'ingénierie inverse. Une prévisualisation des objets qui seront récupérés nous permet de valider les choix et d'exécuter la génération du MLD et du MPD. Après avoir disposé les tables obtenues sur un diagramme, nous obtenons ce qui est présenté en Figure 4. La représentation logique de la base de données permet déjà d'y voir plus clair ! Vous remarquerez que, dans Visual Paradigm, les MLD et MPD se regroupent dans un seul type de diagramme appelé « Entity Relationship Diagram ». Ce diagramme utilise la notation graphique de Bachman pour ce qui est des relations.

Générer le MCD

Le MLD obtenu offre déjà une vision simplifiée de la base de données. Pour le concepteur qui s'intéresse à la structure des données, cette vision peut encore l'être davantage. En effet, il est possible de faire abstraction, par exemple, des colonnes de clés étrangères qui sont propres à la technologie des bases de données relationnelles. C'est l'ingénierie inverse du MLD vers le MCD qui permet cela. L'automate de transformation MVC-CD est nécessaire cette fois, car Visual Paradigm ne prend pas réellement en charge la notion de modèle conceptuel. Lors de l'exécution de l'automate, une fenêtre permet de renommer les éléments conceptuels. Cela permet, par exemple, de définir des noms d'entités au singulier. Suite à cela, l'automate crée les éléments. La représentation utilisée pour le modèle conceptuel est



4 *MLD/MPD obtenu par reverse engineering avec Visual Paradigm*

le diagramme de classes UML. En disposant convenablement les éléments obtenus, nous parvenons au résultat illustré en Figure 6.
Vous remarquerez la richesse du modèle qui permet de faciliter la lecture de la structure de données ! Nous pouvons facilement lire que l'entité « Fourniture » est une spécialisation de « Article », ce qui explique les cardinalités maximales à 1 que nous voyions aux extrémités de la relation située entre les 2 tables correspondantes sur le modèle logique en Figure 4. Nous remarquons également que « Offre » est une entité associative utilisant la représentation de la classe associative UML. Les stéréotypes « UID-1 » indiquent des identifiants uniques, alors que « M » pour « Mandatory » signale un attribut obligatoire.

Illustration avec Oracle Data Modeler

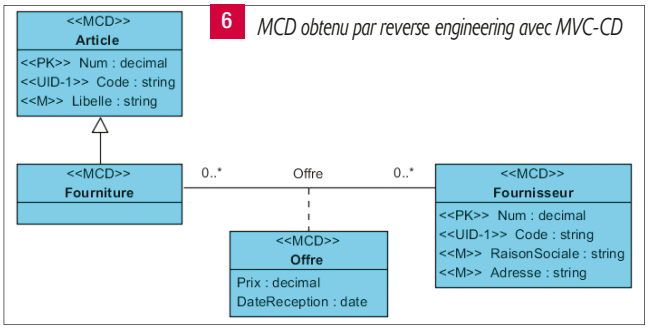
Environnement de développement mis à disposition gratuitement par Oracle, SQL Developer offre une fonctionnalité appelée Data Modeler qui permet de générer une base de données à partir de modèles, ainsi que de réaliser de l'ingénierie inverse. Cet outil, contrairement à Visual Paradigm, est adapté avant tout aux bases de données Oracle, bien qu'il supporte également SQL Server et DB2. Par soucis de simplification, nous emploierons le terme raccourci « Oracle Data Modeler » dans la suite de cet article.

Générer le MLD et le MPD

Pour générer les modèles logiques et physiques à partir d'une base de données avec Oracle Data Modeler, il faut se rendre dans le menu « Fichier » et sélectionner l'option « Data Modeler » puis « Importer » et enfin « Dictionnaire de données ». Un assistant nous guide et nous permet de choisir les objets à récupérer. Une fois les étapes terminées, les MLD et MPD sont créés. Seul le MLD est visible sur un diagramme. Après un réarrangement graphique des tables sur le diagramme, nous obtenons ce que vous pouvez visualiser en Figure 7.
La représentation graphique utilisée ici diffère de celle utilisée par Visual Paradigm. Les relations sont affichées à l'aide de flèches, l'extrémité de la flèche indiquant une cardinalité maximale à 1. Par défaut, les contraintes de clé primaire, de clé étrangère, d'unicité ainsi que les index sont affichés. Ces éléments peuvent être masqués, ce que nous avons effectué ici pour les index de façon à alléger quelque peu le diagramme.

Générer le MCD

Oracle Data Modeler prend en charge la notion de modèle conceptuel, aussi bien pour générer un modèle logique que pour effectuer une ingénierie inverse ! L'ingénierie inverse du MLD en MCD se réalise à l'aide d'une icône accessible lorsque le modèle logique est ouvert. Oracle Data Modeler nous permet, à l'aide d'un glossaire, de définir des termes avec leur orthographe au singulier et au pluriel, ce qui lui permet d'utiliser des noms singuliers pour créer les entités, en se basant sur les noms des tables au pluriel. La Figure 8 vous présente le résultat obtenu. La notation utilisée



6 *MCD obtenu par reverse engineering avec MVC-CD*

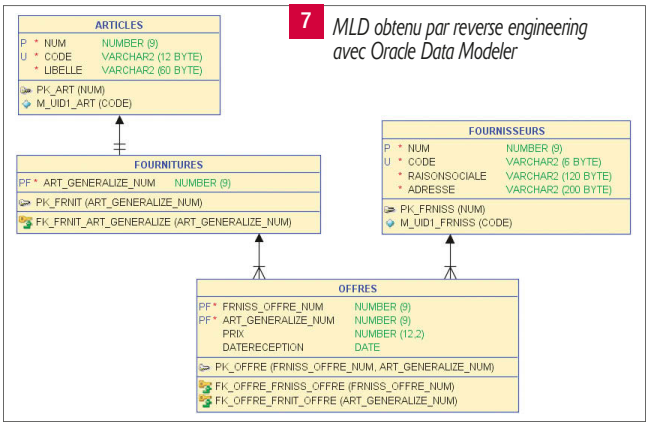
vient de Barker. La lecture des cardinalités est particulière. En effet, la cardinalité minimale se lit du côté de l'entité, alors que la cardinalité maximale se lit du côté opposé de l'entité. Le modèle conceptuel obtenu avec Oracle Data Modeler est épuré de tout élément technique. Ici également, il nous semble évident que la lecture du MCD est plus facile que celle du MLD ! Notons cependant que la représentation Oracle Data Modeler est moins riche que celle du diagramme de classe UML. Par exemple, le sens de l'entité « Fourniture » n'est pas compréhensible de manière aussi évidente que sur le diagramme UML de la Figure 6, où le lien de spécialisation montre clairement que la fourniture est une sorte d'article.

CONCLUSION

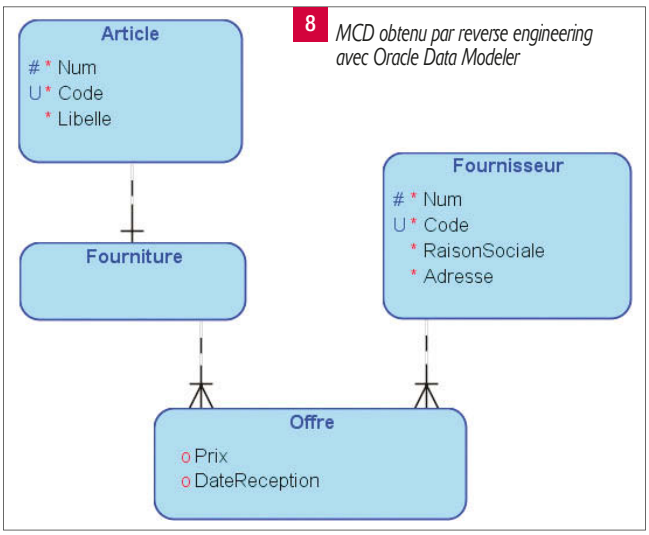
Vous connaissez maintenant le concept de l'ingénierie inverse à partir d'une base de données et comprenez les différents usages qui peuvent en être faits. Avec les illustrations réalisées au travers de deux outils différents, vous avez pu remarquer qu'il existe des différences, notamment en matière de notation graphique, mais vous avez aussi vu que le concept général reste le même. Au vu du peu d'investissement que le processus d'ingénierie inverse nécessite et du gain qu'il apporte, nous ne pouvons que vous conseiller de l'utiliser dans votre environnement, autant pour créer ou mettre à jour des modèles servant de documentation, que pour faire évoluer une base de données par réingénierie ! •

Références

Projet MVC-CD et téléchargement de l'automate de transformation : http://lgl.isnetne.ch/Sagex35793/index.htm
Droz-dit-Busset Michaël que nous remercions, Travail de Bachelor « MVC-CD – Ingénierie inverse », juillet 2015



7 *MLD obtenu par reverse engineering avec Oracle Data Modeler*



8 *MCD obtenu par reverse engineering avec Oracle Data Modeler*

Service Fabric : une plateforme PaaS sous stéroïdes Partie 1

• Maxime CAROUL
mcaroul@infinitesquare.com
blogs.infinitesquare.com
• Thibaut RANISE
tranise@infinitesquare.com
INFINITE SQUARE

Derrière ce titre quelque peu marketing se dresse un constat simple, l'arrivée d'une nouvelle génération de solutions Platform as a Service (PaaS) qui offre davantage de performance, de souplesse et d'évolutivité à nos applications. Cet article a pour objectif de vous présenter les avantages et inconvénients de cette offre avec, aussi souvent que possible, un élément de comparaison ou une mise en situation pour mieux comprendre le PaaS v2.

Azure Service Fabric c'est quoi ?

C'est une plateforme middleware qui a été développée par Microsoft et éprouvée en production depuis 2010 pour offrir plusieurs services internes (Cortana, PowerBI, Skype Entreprise, Intune, etc.) et de nombreuses offres Azure (SQL Database, DocumentDb, Event Hubs, Azure IoT, Azure Data Factory, etc.). Il suffit de reprendre quelques chiffres annoncés par Microsoft durant la conférence Build 2016 pour imaginer les capacités de cette solution :

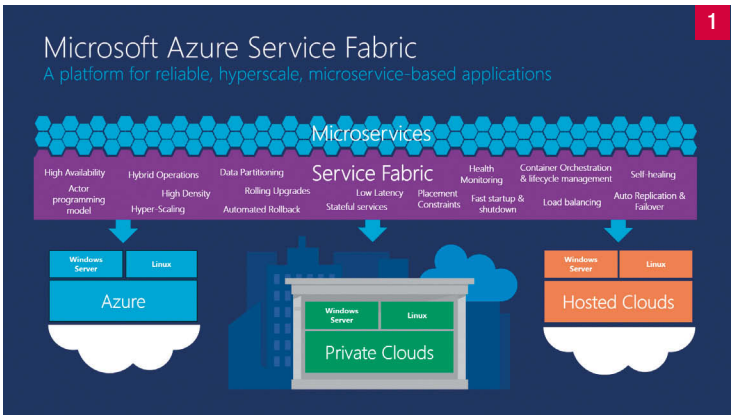
- SQL Database : plus d'1,4 millions de base de données ;
- Azure IoT : 2 milliards de messages traités par semaine ;
- Event Hub : 20 milliards d'évènements par jour.

Annoncée en disponibilité générale durant cette même conférence, elle permet de créer et d'héberger des applications hautement disponibles, évolutives et durables tout en proposant un ensemble d'outils, de bonnes pratiques pour gérer l'infrastructure sous-jacente. En tant que développeur dans le Cloud, il n'y a rien de nouveau puisque ces termes sont déjà utilisés depuis un moment. Avant d'aller plus loin, j'aimerais revenir sur le besoin actuel des entreprises. Parce que celui-ci n'est pas statique, il évolue en même temps que nos métiers. Et cela va me permettre de présenter les services qu'offre Microsoft en réponse. En tant qu'entreprise adepte des solutions PaaS :

- Je souhaite pouvoir démarrer très petit avec par exemple un PoC (Proof of Concept) ou MVP (Minimum Viable Product) tout en ayant la possibilité de grossir très vite en termes de consommation de ressources ;
- Je veux être focus sur mon métier, bénéficier d'outils et d'une bonne pratique pour gérer ma plateforme ;
- Je souhaite optimiser l'utilisation de mes ressources car tout a un prix ;
- Néanmoins je ne compte pas renier sur la fiabilité ou la performance de mes applications ;
- J'exige une grande souplesse concernant la pile technologique et le fournisseur de service afin d'avoir toujours le choix. Qui ne se souvient pas du départ de Dropbox de chez Amazon ?
- Je veux que cela s'inscrive dans une démarche simple et agile.

Voici un rapide résumé de ce que proposait Microsoft avant Service Fabric :

Point	AppService	AppService Environnement	CloudService
Scalabilité	20 instances max en Premium	50 instances	1000 instances
PaaS	Abstraction forte avec l'infrastructure		Plus de contrôle sur l'OS
Densité	Varie en fonction votre architecture		
Performance	Répartition de charge, mise à l'échelle automatique, etc…		
Technologies	PHP, Java, Python, .Net et Node.js		
Simple et Agile	Déploiement continue, slot, routage du trafic de production, backup, webjob, kudu, appservice editor		Accès à distance



1

On constate facilement certaines faiblesses dans ces offres PaaS :

- Couplage fort à Azure (sans parler d'Azure Stack) et Windows ;
- Pas d'évolutivité très importante ou alors dans une offre pas forcément adaptée ;
- Pas de scénario de déploiement avancé ;
- Pas de scénario de test en production avancée ;
- Etc.

Azure Service Fabric fournit une plateforme qui permet de créer des applications hautement évolutives, disponibles et faciles à gérer. Pour cela, un certain nombre de concepts et d'outils est disponible :

- Gestion du cycle de vie des applications ;
- Architecture microservice ;
- Testabilité en production ;
- Simplicité ;
- Haute disponibilité.

Dans la suite de cet article, nous allons voir comment Service Fabric permet de répondre plus efficacement à ces besoins.

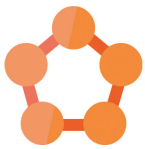
Abstraction entre les ressources et l'infrastructure

L'objectif d'Azure Service Fabric est d'offrir une abstraction entre les ressources (CPU, RAM, Réseau) nécessaires à vos applications et l'infrastructure sous-jacente. C'est pourquoi il peut être déployé dans n'importe quel environnement :

- Machine locale de développement ;
- Machine physique ou virtualisée ;
- Cloud (Azure ou n'importe quel autre fournisseur Cloud en IaaS) ou OnPremise (il est prévu qu'Azure Stack supporte nativement Service Fabric) ;
- Windows (Server 2012 R2 ou 2016) et Linux (en preview limitée à Ubuntu Server 16.04). [Fig.1].

A savoir, dans le cas OnPremise vous êtes responsable de maintenir votre

infrastructure sous-jacente. De même, comme un produit Microsoft est toujours mieux intégré dans un environnement du même nom, la mise à l'échelle automatique n'est disponible qu'avec un environnement sur Azure. Maintenant voyons les bases : pour héberger vos applications, Service Fabric gère un cluster qui est un ensemble de nœuds. Chaque nœud est un environnement d'exécution pour vos services. Dans un scénario classique, un nœud correspond à une machine (virtualisée ou physique). Une illustration simpliste d'un cluster est représentée dans son logo. Il s'agit d'un ensemble de nœuds égaux arrangés en cercle. En réalité, un nœud est capable de communiquer directement avec chacun d'entre eux et non uniquement ses voisins proches. De plus, un nœud peut avoir des responsabilités plus importantes (Cf. partitionnement).



« Un cluster Service Fabric est capable de gérer des nœuds avec différentes capacités matérielles. »

Agilité via les microservices

Service Fabric est une plateforme qui héberge des services dits « nés dans le Cloud », c'est-à-dire qu'ils peuvent commencer petit puis, si nécessaire, évoluer à grande échelle avec plusieurs milliers d'instances.

Cette flexibilité est due au modèle micro-services qui a pour but de décomposer une application monolithique en plusieurs services indépendants et autonomes. Son usage facilite la maintenance applicative tant au niveau développement qu'opérationnelle. En effet, le couplage faible entre les différents services rend toute modification plus simple et maîtrisée. Côté mise à jour, on passe d'un mode tout-en-un à une gestion incrémentale ciblée uniquement sur le type de service modifié ce qui facilite la haute disponibilité.

« Même s'il y a peu d'intérêt à ne pas l'utiliser, le modèle micro-services n'est pas obligatoire.. »

Concrètement un micro-service peut prendre la forme d'une application ASP.NET, Node.js, Java ou tout fichier exécutable (.exe).

Qualité de service

Dans tout système distribué, il est impossible de garantir en même temps les trois contraintes suivantes : consistance des données, disponibilité des services et tolérance à la panne (Wikipédia : théorème CAP). Malgré le fait que ce théorème soit établi, cela n'empêche pas Service Fabric d'offrir un niveau satisfaisant pour les deux premiers. Le dernier étant non négociable pour une entreprise.

Pour ce faire, Service Fabric offre une complète abstraction entre les ressources (CPU, RAM, Réseau, etc…) dont vos services ont besoin à l'instant T et votre infrastructure. En d'autre termes, la plateforme peut décider de déplacer vos services d'un nœud à l'autre si la charge appliquée sur le nœud actuel est trop importante. Ce mécanisme associé aux micro-services permet d'offrir une haute densité d'instance de service par nœud et donc d'optimiser au mieux les ressources.

```
PS C:\WS\ServiceFabric> tree /f
Folder PATH listing
Volume serial number is 000000B9 6C58:676D
C:.
    ServiceFabric.sln

    CalculatorApp
        CalculatorApp.sfproj
        packages.config

        ApplicationPackageRoot
            ApplicationManifest.xml

        ApplicationParameters
            Cloud.xml
            Local.xml

        bin
        obj
        PublishProfiles
            Cloud.xml
            Local.xml

        Scripts
            Deploy-FabricApplication.ps1

    CalculatorService
        App.config
        CalculatorService.cs
        CalculatorService.csproj
        packages.config
        Program.cs
        ServiceEventSource.cs

        bin
        obj
        PackageRoot
            ServiceManifest.xml

            Config
                Settings.xml

        Properties
            AssemblyInfo.cs
```

3

Modèle d'application

Pour héberger et gérer vos applications, Service Fabric impose une structure. Il est important de la présenter car cela permet mieux comprendre le fonctionnement de la plateforme. [Fig.2].

Une application regroupe un ou plusieurs services. Chaque service est une unité d'exécution complète et autonome. Il est constitué de trois éléments : du code (fichier binaire exécutable), une configuration (paramètres) et des données statiques.

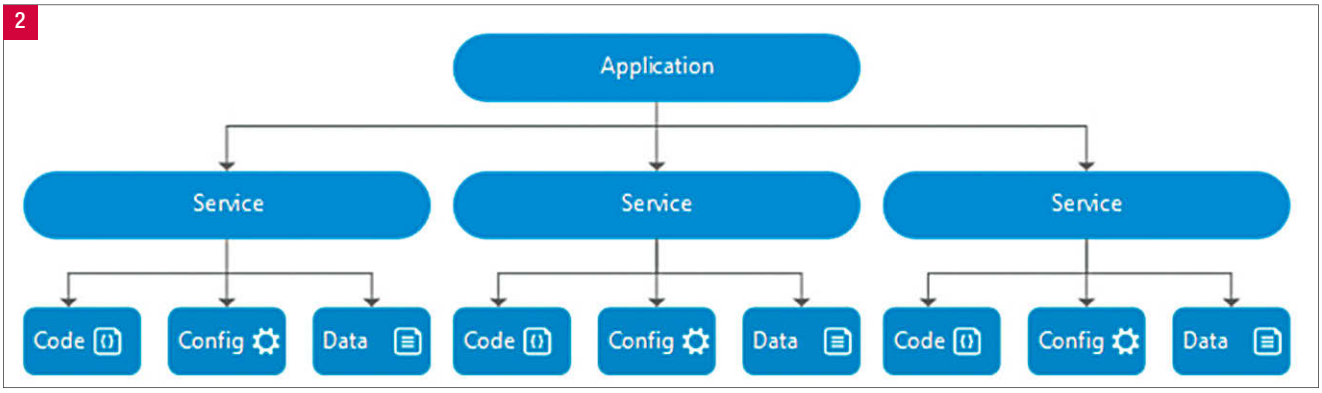
Voyons maintenant à quoi ressemble la structure d'un projet d'application Service Fabric composé d'un service. [Fig.3].

On remarque la présence d'un projet pour l'application et un autre pour le service. Chaque projet contient un fichier manifeste qui référence entre autres les éléments associés.

Ainsi, le fichier ApplicationManifest.xml liste les services qui constituent l'application et le fichier ServiceManifest.xml définit les trois éléments d'un service. Sachez que chaque élément est défini par un nom et une version. •

Suite dans Programmez n°205.

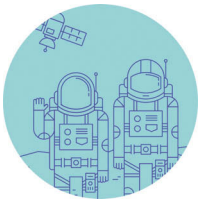
2



Les **Progressive** Web Apps

Partie 2

La seconde chance du Web dans l'univers du mobile



SIMON GOMBAUD
Directeur artistique
Made on Mars

Au milieu de l'année 2007, voilà déjà presque 10 ans, une révolution se met en place avec l'arrivée du tout premier iPhone. Les utilisateurs ont un nouveau terminal avec lequel ils peuvent naviguer sur Internet. À l'époque, le Web n'est pas préparé à cette révolution. Les sites ont encore une majorité d'éléments en Flash qui ne sont pas supportés par l'iPhone. Steve Jobs annoncera d'ailleurs 3 ans plus tard dans sa célèbre lettre ouverte "Thoughts on Flash" qu'il n'a aucunement l'intention de changer ça. Été 2008, Apple dévoile l'iPhone 3G et l'App Store.

Sécurisée

Pour utiliser les services workers, nous sommes contraints de servir notre PWA via HTTPS afin de sécuriser les données des utilisateurs. La configuration d'un domaine en HTTPS m'a toujours semblé être un long parcours du combattant semé de certificats et autres clés obscures. De plus, obtenir le précieux certificat a un coût qui peut aller de quelques euros à plusieurs centaines d'euros.

Let's Encrypt

Mais ça c'était avant. Depuis un peu plus d'un an, le projet Let's Encrypt https://letsencrypt.org a vu le jour, soutenu par les plus grandes sociétés de l'Internet visant à automatiser tout le processus de création, signature et validation des certificats.

Let's Encrypt fournit un outil appelé Certbot https://certbot.eff.org . Une fois installé, une seule commande est nécessaire pour générer votre certificat.

```
letsencrypt certonly --webroot -w /var/www/example -d example.com -d www.example.com -w /var/www/thing -d thing.is -d m.thing.is
```

Les certificats générés par Let's Encrypt sont valables pendant 90 jours. Mais ne vous inquiétez pas, ils ont intégré une commande pour les renouveler tous en même temps.

```
letsencrypt renew
```

Il suffira de créer un cron job pour appeler cette commande et le tour est joué.

Peut être installée

Sur la toile, installer une Web app revient à la mettre en favoris en quelque sorte. L'idée est d'être à un seul clic du contenu. Sur mobile, l'ergonomie tactile est telle que l'accès aux favoris demande déjà une certaine expertise. Mais aujourd'hui, grâce au manifest.json, on peut encourager l'installation de l'application via une bannière permettant ainsi une expérience plus simple pour les moins initiés. On ajoute un fichier manifest.json dans notre dossier public/ avec quelques lignes de configuration. Le déclenchement de la bannière d'installation (appelée web app install banners) est entièrement géré par Google Chrome. Son affichage dépend des critères suivants : Le manifest.json doit avoir :

- Le paramètre short_name, correspondant au nom de l'application dans l'écran d'accueil ;
- Le paramètre name, correspondant au nom utilisé dans la bannière d'installation ;
- Une image d'icône de taille 144x144 au format png ;
- Le paramètre d'URL de démarrage start_url doit être spécifié.

L'application doit aussi :

- Avoir un service worker enregistré ;
- Utiliser le protocol HTTPS (qui est nécéssaire pour le SW de toute manière) ;
- Être visitée au moins deux fois, avec au moins 5 minutes entre chaque visite.

Voici le code et le résultat sur notre application d'exemple.

```
{
  "name": "Love, love, love",
  "short_name": "Love",
  "icons": [
   {
    "src": "images/icons/icon-android-144x144.png",
    "sizes": "144x144",
    "type": "image/png"
   }],
  "start_url": "/"
}
```

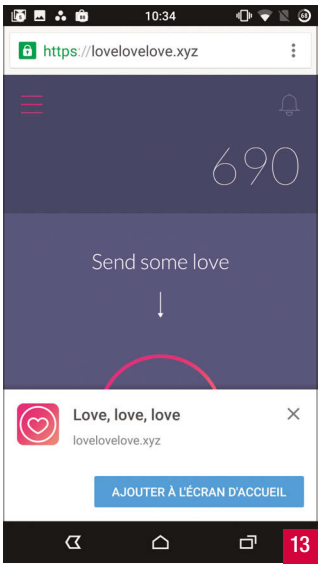
[Fig.13]

Peut être partagée

En 2016, tout le monde partage sur les réseaux sociaux. Ça n'a l'air de rien mais l'impact sur le trafic des sites est énorme. La part de trafic provenant des réseaux sociaux est en effet en constante évolution. Mais pour qu'un lien soit cliqué, il faut qu'il soit attractif.

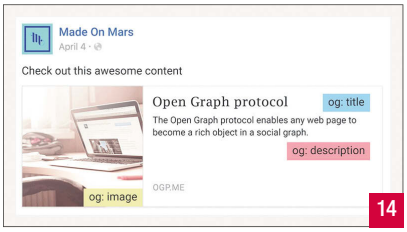
Les balises meta Open Graph

Le Protocole Open Graph permet de donner aux réseaux sociaux (Facebook, Twitter, Google+…) des informations sur votre page. De cette manière, le réseau social ciblé sera capable d'afficher un lien "enrichi" avec un titre, une description et une image : [Fig.14]



Impact sur le SEO

Les balises Open Graph s'appliquant aux réseaux sociaux, elles n'ont en théorie aucu-



ne incidence sur le référencement du site. Cependant, les réseaux sociaux ont pris une telle ampleur ces dernières années que les moteurs de recherche ne peuvent pas ignorer les données des balises. Comme pour les meta title et description, les meta Open Graph sont autant de renseignements supplémentaires sur votre page. C'est donc assurément une bonne pratique à adopter, qui sera certainement récompensée par Google dans le futur.

Mise en place

Comme toutes les balises meta, les balises Open Graph se placent dans le head de votre site :

```
<!DOCTYPE html>
<html lang="fr">
 <head>
  <meta charset="utf-8">
  <title>Love, love, love</title>
  <meta name="description" content="A simple Progressive Web App to send some love">

  /* Meta Tags pour Facebook, Google+, LinkedIn... */
  <meta property="og:url" content="https://lovelovelove.xyz/">
  <meta property="og:site_name" content="Love, love, love | A simple Progressive Web App to send some love">
  <meta property="og:title" content="Love, love, love">
  <meta property="og:description" content="A simple Progressive Web App to send some love">
  <meta property="og:image" content="https://lovelovelove.xyz/img/love-love-love-share.jpg">
  <meta property="og:image:type" content="image/jpg">
  <meta property="og:image:width" content="1200">
  <meta property="og:image:height" content="630">

  /* Meta Tags pour Twitter */
  <meta property="twitter:site" content="https://lovelovelove.xyz/">
  <meta property="twitter:creator" content="Made On Mars">
  <meta property="twitter:card" content="summary_large_image">
  <meta property="twitter:title" content="Love, love, love">
  <meta property="twitter:description" content="A simple Progressive Web App to send some love">
  <meta property="twitter:image:src" content="https://lovelovelove.xyz/img/love-love-love-share.jpg">
  <meta property="twitter:image:width" content="1200">
  <meta property="twitter:image:height" content="630">
 </head>
 <body>
  ...
 </body>
</html>
```

Pour la plupart des réseaux sociaux (Facebook, Google+, LinkedIn et beaucoup d'autres), ce sont les balises Open Graph <meta property="og:…" content="…"> qui sont utilisées. Twitter possède quant à lui ses propres balises spécifiques : <meta property="twitter:…" content="…">. Vous trouverez l'ensemble des spécifications techniques sur le site officiel de l'Open Graph : http://ogp.me/

Référencée

Lors de la réalisation d'une PWA, la question de l'indexation se pose. On pourrait penser que Google, encourageant le développement des PWA, décide de les mettre en avant dans ses résultats de recherche. On l'a vu plus haut, c'est ce qui s'est passé pour les sites responsives ou encore pour les URL en https. Mais à ce jour, ce n'est pas aussi simple.

Pour commencer, il faut savoir que Google analyse les PWA comme un site AJAX ou JavaScript. Googlebot interprète parfois le Javascript, mais ne supporte par les Services Worker. John Mueller, porte-parole SEO de Google, a publié en mars 2016 un article détaillant les recommandations pour garantir une bonne indexation d'un site JavaScript ou d'une PWA : (https://plus.google.com/u/0/+JohnMueller/posts/LT4fU7kFB8W).

Suivant la même logique d'optimisation que le récent format AMP (Accelerated Mobile Pages - https://www.ampproject.org/), les PWAs pourraient demain bénéficier d'une mise en avant dans les résultats de recherche, avec les données issues des balises Open Graph, ou du manifeste JSON.

La suite dans Programmez ! 205

MADE ON MARS

Made On Mars est une agence digitale rennaise fondée en 2016 par Thomas Foricher (directeur technique) et Simon Gombaud (directeur artistique). Passionnés de digital, ils conçoivent des applications mobiles et des sites Web pour aider les entreprises à développer leur e-réputation. Ils proposent du conseil en stratégie digitale, du développement et de la création graphique. https://www.made-on-mars.com/fr/

Découvrir GitHub

Partie 2

Paquet Judicaël
DSI chez Batiwiz, Judicaël est spécialiste dans le coaching agile et l'architecture logicielle/devops

Devenu un incontournable du monde de l'open source, GitHub est une plateforme de contrôle de version et de collaboration basée sur l'excellent Git, gratuit pour tous développeurs souhaitant contribuer à leur manière au monde open source.

Le droit à l'erreur

Git vous autorise de vous tromper et vous propose une ligne de commande fort utile pour remettre votre code au dernier commit

```
git reset HEAD
```

Nous pouvons également revenir au commit précédent comme ceci :

```
git reset HEAD^
```

En ajoutant encore un ^, vous pourrez remonter encore d'un commit supplémentaire. Si vous mettez le numéro de commit à la place du HEAD, vous pourrez mettre votre code directement au commit concerné. Vous pouvez également annuler tous les changements du dernier commit en faisant la commande ci-dessous, mais il faut la manipuler avec une extrême précaution car l'action est irréversible :

```
git reset --hard HEAD^
```

Si vous voulez remettre un fichier dans son état initial (du dernier commit) car au final vous ne voulez plus des modifications apportées, vous pouvez faire ceci :

```
git checkout monfichier
```

Lorsque vous avez ajouté un fichier avec un git add que vous ne voulez au final pas ajouter, vous pourrez supprimer celui-ci en faisant ceci :

```
git reset HEAD -- monfichier
```

Cependant, si vous avez déjà fait le commit que vous désirez annuler, vous pouvez également le faire mais avec une autre méthode :

```
git revert numero_commit
```

Changer de branche sans commit

Le titre peut faire sourire mais lorsqu'on change de branche, on ne veut pas forcément que les modifications en cours restent, mais qu'elles ne soient conservées que sur la branche en cours.
Une commande Git permet justement de faire ça et évite de tenter d'autres opérations pour en arriver à ses fins :

```
git stash
```

Lorsque vous reviendrez sur votre branche, vous pourrez récupérer ces fameuses modifications conservées en faisant la commande suivante :

```
git stash apply
```

J'espère que l'ensemble des petites astuces fournies sur cette page vous seront utiles à l'avenir.

Connaître les branches distantes

Voici une méthode fort utile pour prendre connaissance des branches distantes disponibles :

```
Git fetch
git branch -va
```

Nouvelle branche à partir d'une branche distante

Il est parfois bien utile de créer une nouvelle branche à partir d'une autre branche et pas forcément à partir du master :

```
git checkout -b hotfix-monbug origin/hotfix-monbug
```

Faire du GitHub avancé

Github propose des notions encore plus avancées que ce que nous venons de voir. Afin de faire de vous des experts des communautés open source, nous allons les voir dès maintenant.

Forker un projet

Github permet soit de forker un projet, soit de travailler sur un projet directement de son Github. Pour réaliser des commit/push sur un dépôt, il faut soit en être propriétaire, soit en être collaborateur, ce qui ne sera pas forcément le cas. Cependant, si vous avez une modification importante à proposer à l'auteur, il vous suffit d'aller sur le dépôt de celui-ci et de cliquer en haut à droite sur « Fork ». Le projet sera ensuite bien dans un nouveau dépôt sur votre compte Github.
Un dépôt forké aura quelques particularités comme rappeler que le dépôt est le fork d'un autre projet (sous le nom du projet) et vous permettra de proposer un pull request au projet initial de votre dépôt (sur la branche désirée).
Ce procédé est très populaire car un projet tel que nodejs par exemple a été forké 4303 fois, ce qui montre que certaines communautés sont assez gigantesques.

Un peu de sécurité sur mon compte

Github est vraiment un outil complet car il permet également de surveiller votre compte. Si vous allez sur votre icône en haut à droite, ensuite dans « settings », puis le menu « security », vous pourrez voir les sessions en cours avec quelques informations utiles pour voir si vous êtes bien la seule personne à utiliser votre compte (voire sur quels ordinateurs). Vous pouvez également voir l'historique de ce qui se passe (actions des collaborateurs sur vos dépôts, les actions des logiciels tiers que vous avez accepté d'actionner sur votre github ainsi que des actions courantes comme des tentatives de connexions échouées).
Pour une raison X, vous avez également la possibilité de bloquer des actions d'un utilisateur dans le menu « blocked user ».
Comme pour facebook connect, vous pourrez visualiser tous les outils que vous avez autorisés à se connecter à vos dépôts Github dans le menu « Authorized applications » ; si vous ne désirez plus voir une application se connecter sur votre Github, vous pourrez la supprimer à partir de ce menu (pour l'ensemble des dépôts).

Faire de l'intégration continue (CI)

Github propose de se connecter avec différents outils d'intégration continue afin de vous donner la possibilité de mettre en place toute une chaîne automatisée entre le développeur et la production.
Vous pouvez par exemple ajouter Travis-CI qui est à ce jour le plus utilisé, et totalement gratuit sur vos dépôts publics : https://travis-ci.org/. Vous pourrez lancer du test unitaire avec phpunit par exemple ; il validera ou non vos push. Allez sur le site de Travis et faites un « sign up with Github ». Dès que c'est bon, allez dans « account » et sélectionnez les dépôts que vous désirez mettre en test.
Pour que Travis fonctionne, il faudra rajouter un fichier .travis.yml à la racine de votre dépôt :

```
language: php
php:
  - '7.0'
before_script:
  - composer install

script:
  - phpunit --coverage-text
```

Vos dépôts seront ainsi régulièrement testés et chaque commit se verra attribué d'une croix rouge indiquant que le test a échoué, ou d'une coche verte si les tests sont passés avec succès. Vous pourrez aller plus loin avec l'outil, comme faire du déploiement, mais je vous laisserai approfondir ce sujet par vous-même.

Connecter Slack et Github

Slack qui est à ce jour un des outils les plus utilisés et principalement dans le monde des startups propose une option pour se connecter à Github. Vous pourrez lui dire d'ajouter des notifications au sein d'un channel choisi selon le type d'évènements désirés : commit/push, nouveau commentaire, de pull request… Un petit plus qui renforce le travail communautaire.
Pour configurer cela, allez dans app & intégration, cherchez « github » et suivez les instructions.

Et mon application ?

En tant que développeur, vous pouvez également proposer un nouvel outil d'aide aux développeurs qui se connecterait à Github en allant dans « settings » puis dans les onglets de « developer settings ».
Je n'aurai pas la place dans ce numéro de vous expliquer comment faire cela, mais si cela vous intéresse, une documentation est disponible sur Github pour vous aider. A présent, Github n'a plus de secret pour vous à part peut-être de ne pas connaître toutes les applications externes qui s'y connectent et qui peuvent vous aider à consolider encore plus votre environnement dit Devops.

Créer un paquet Composer

Qu'est-ce que Composer

Composer est un gestionnaire de paquets très utilisé dans le monde PHP au même titre que npm pour node.js et apt-get pour Debian/Ubuntu. Il permet d'installer des applications ou librairie en automatisant l'installation des dépendances.

Préparation du dépôt

Je vais commencer par créer un fichier PHP dans mon dépôt test comme ceci :

```
<?php

declare(strict_types=1);

namespace monNamespace ;

function test (): int {
   return 1;
}

echo test();
```

Le contenu de ce code n'est pas très important mais sachez que vous êtes obligé d'être en PHP 7 pour pouvoir le lancer sans erreur.
Pour créer un package Composer valide, il faudra créer un fichier composer.json à la racine de votre dépôt qui ressemblera à ceci :

```
{
 "name" : "[pseudo]/test",
  "description" : "ma description",
  "type" : "library",
  "version": "1.0.0",
  "authors" : [{
     "name" : "Judicaël Paquet",
     "email" : "judicael.paquet@gmail.com"
  }],
  "keywords" : ["lzberg", "bundle"],
  "license" : ["MIT"],
  "require" : {
     "php": ">=7.0.0"
  },
  "autoload" : {
     "psr-0" : {
        "monNamespace" : ""
     }
  }
}
```

N'oubliez pas de remplacer [pseudo] par votre pseudonyme. Vous pouvez spécifier un autre nom de package (ici 'test') que le nom du dépôt Github. Ici, nous avons bien indiqué que notre code exige que php soit au minimum la version 7.0.0 car nous avons vu que le code ne fonctionnerait pas sur les précédentes versions.
Vous faites un commit et un push de votre code pour le mettre sur votre dépôt test de votre Github.

Création de compte packagist

Packagist est la plateforme de dépôt PHP open source principal du monde PHP sur lequel se base Composer : https://packagist.org/.
Créez votre compte (avec votre connexion Github ça sera parfait) et autorisez celui-ci à utiliser vos données publiques. Packagist est un site de confiance, n'ayez pas peur de valider.
Maintenant que vous êtes connecté, cliquez sur « submit » en haut de page et mettez l'URL de votre dépôt Github dans le formulaire https://github.com/[pseudo]/test avant de cliquer sur « submit ».

Comme dirait l'un des développeurs de mon équipe, sous vos yeux émerveillés, voilà votre package publié dans le monde de l'open source.

Et si on automatisait ?

Github prend encore plus de puissance à ce niveau-là car il propose des 'webhook' qui vont automatiser les envois de versions à packagist pour offrir à la communauté vos améliorations sans le moindre effort. Si tout se passe bien, vous n'irez sur packagist qu'en tant que visiteur ensuite.

Toujours sur Packagist, allez à [pseudo] en haut à droite puis profil. Sur cette nouvelle page, vous pourrez récupérer un token indispensable en cliquant sur « show token api » qui va être la sécurité pour connecter Github et Packagist ensemble.

Sur la page principale de mon dépôt Github, je vais aller dans l'onglet « Settings », ensuite « Integration & services » et je vais ajouter un nouveau service packagist (voir [Fig.4]).

Vous entrez ensuite le login, le token packagist et le domaine packagist.org avant de valider. Vous pouvez tester la connexion de votre service avant de valider ci-besoin.

Pour tester l'automatisation de la mise à jour de votre paquet, modifiez votre composer.json en mettant "version" : "1.0.1" et faites un commit/push du fichier.

Sur votre dépôt Github, allez dans release et créez une nouvelle release avec le tag 1.0.1 @ master (vous pouvez publier sans remplir le reste du formulaire).

Faire une GitHub Page

Générer à partir d'un dossier docs/

Github vous permet de créer un petit site Web qui accompagnera votre projet afin de mieux le marketer, ainsi que de proposer des documentations en ligne. En indiquant la source sur docs/ de votre dépôt, Github exposera vos pages HTML directement en ligne (une URL vous est fournie). Par défaut l'URL de votre dépôt sera sur le domaine : https://[utilisateur].github.io/[depot]/ (ou organisation à la place de l'utilisateur).

A présent, vous pourrez gérer votre site Web en modifiant directement les fichiers du dossier docs/ de votre projet. Comme pour un site Web classique, le fichier index.html est le fichier lancé par défaut.

Créer un site à partir d'un template

Github vous permet également d'avoir un joli site Web (dans ce même menu Options) pour votre projet en vous proposant de choisir un template défini. La gestion de ce site est un peu différente car le template a créé une nouvelle branche gh-pages qui contient le contenu de base de ce site. Vous pourrez donc travailler votre nouveau site en travaillant directement sur cette branche de votre dépôt. Les puristes n'aimeront sûrement pas cette magouille, mais cette façon de faire reste pratique à mettre en œuvre. De plus, le code de votre contenu contrairement à la première méthode ne sera pas envoyé inutilement aux utilisateurs de votre projet. Comme pour la première méthode proposée, vous aurez accès à ce site par cette URL https://[utilisateur].github.io/[depot]/ (ou organisation à la place de l'utilisateur).

Créer un site à partir d'un template

L'outil propose une option encore plus avancée pour définir son propre sous-domaine sur ce site Web en ajoutant une ligne dans votre zone dans la gestion de votre DNS :

```
doc 3592 IN CNAME vyctory.github.io.
```

Ici on proposerait à doc.mondomaine.com d'atteindre directement notre documentation. Dès que cela est fait, allez confirmer sur cette même page de gestion Github le sous-domaine que vous venez de créer afin qu'il soit accepté de leur côté.

Pour aller plus loin

Un joli site Web présente GitHub Page si vous désirez approfondir le sujet : https://pages.github.com/. Les possibilités sont multiples et vous pourrez créer un petit site complet sur votre projet.

Sachez que vous pouvez forcer l'HTTPS sur ces pages (aujourd'hui c'est une exigence Google pour le SEO) et votre site Web sera en responsive design pour permettre aux visiteurs mobiles de bien voir votre site.

Des outils utiles

Comme vous venez de le voir, nous pouvons connecter Github avec de nombreux outils externes. Avant de refermer cet article qui je l'espère a su vous montrer toute la puissance de Github, voici quelques outils que vous pourriez utiliser dans votre usine logicielle :

AWS CodeDeploy : permet de déployer du code sur vos serveurs AWS (Amazon) de façon automatisée.

Sensiolabs Insight : permet une analyse complète du code PHP (voire de Symfony ou Drupal) pour vous donner des conseils d'optimisation de qualité. Une note sera attribuée à chaque analyse pour que vous sachiez vous situer au niveau qualité du code.

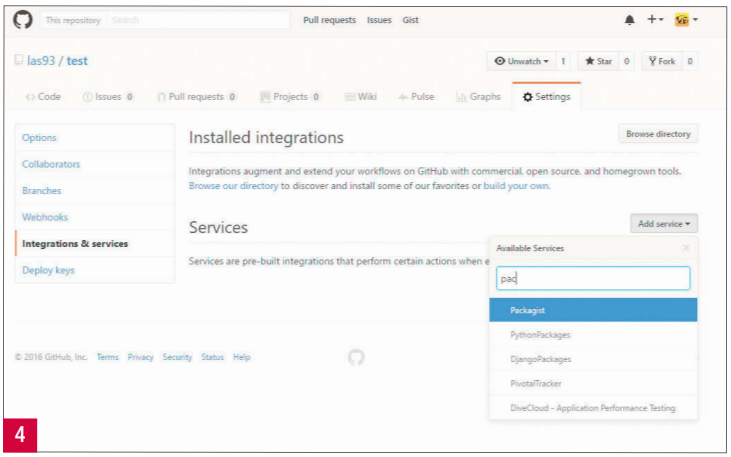
Trello : outil de gestion agile (kanban principalement) qui peut se connecter avec Github. Très pratique pour lier ingénierie logicielle et organisation agile des équipes.

Scrutinizer CI : autre outil d'analyse de code permettant d'améliorer son code. Les axes d'amélioration sont différents de Sensiolabs Insight et apporte également son intérêt ; il est disponible également pour d'autres langages.

ET POUR CONCLURE

Même si on ne le voit pas forcément au premier abord, Github est une plateforme très complète qui peut vous permettre de mettre en place une usine logicielle complète sachant que les connexions multiples avec des logiciels tiers sont nombreuses.

Un peu comme un Aloglia pour la recherche, Github peut être la base d'une intégration continue (CI) gérée intégralement en ligne comme le montre cet article. Ce n'est pas si compliqué et c'est probablement un très bon début pour se lancer dans des sujets comme le travail collaboratif ou l'intégration continue. •

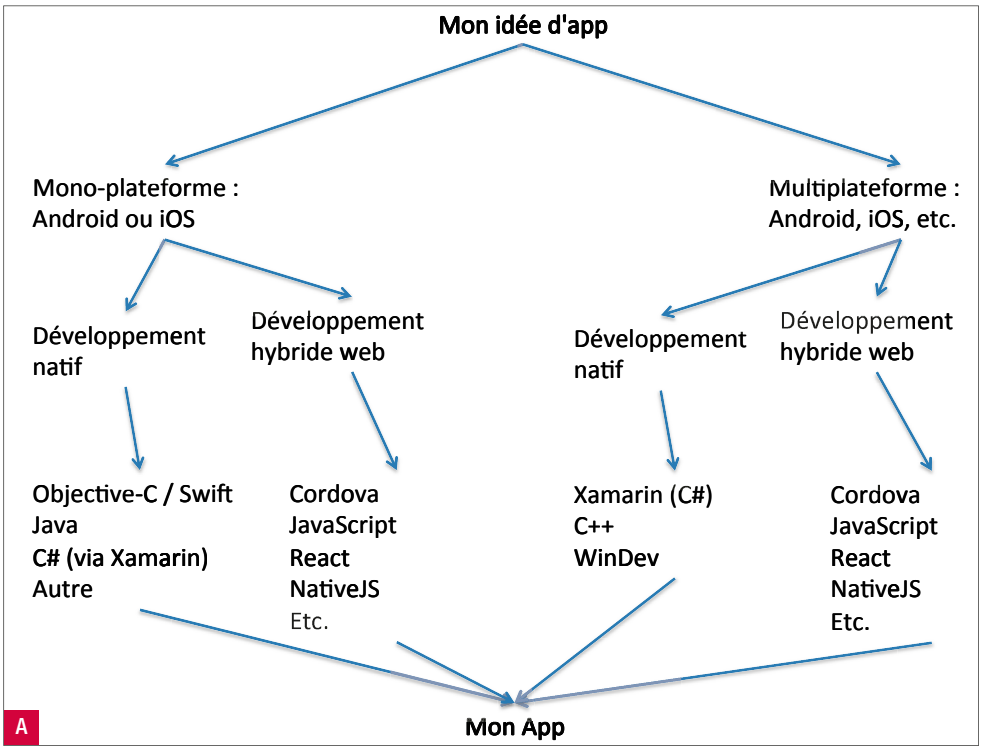


4

Développement mobile : choisir, et si possible bien choisir

En 10 ans, le développement mobile a totalement changé que ce soient les matériels, les systèmes et les outils de développement. Au début, nous étions en natif, natif et encore natif. Puis nous sommes passés à l'hésitation multiplateforme avec des outils plus ou moins fiables, puis aux apps Web. Aujourd'hui, où en est-on ?

La rédaction



A

Le développeur, qu'il soit débutant ou expert, n'a que l'embarras du choix, et c'est bien là le problème. Quel outil choisir ? Schématiquement, nous pouvons résumer le premier niveau de réflexion à un diagramme très simple : [A]

En y réfléchissant un peu, nous allons vite nous apercevoir que de nombreuses questions se posent derrière et qu'elles sont tout aussi fondamentales :

- Est-ce un développement "loisir" ou un projet de grande envergure ?
- Est-ce une app déployée sur les Apps Stores pour tout le monde ou une app plutôt interne à l'entreprise ?
- Quelles sont mes compétences ?
- Quel budget ai-je à ma disposition ? Faut-il investir dans des formations, des programmes développeurs ou dans les outils ?
- De quelles fonctions ai-je besoin dans l'app ? Est-ce que je dois beaucoup utiliser le matériel ou non ?
- L'interface est-elle très spécifique ou très généraliste ?
- Mon app est-elle monolangue ou multilangue ?
- Est-ce que je travaille seul ou en équipe ? Dans ce cas, quelle gestion de projets faut-il mettre en place ?
- Quels sont les tests terminaux à faire ? Et avec quels outils ?
- L'app aura-t-elle un cycle de vie régulier (mise à jour, maintenance, etc.) ?
- Me faut-il un backend et des services backend spécifiques (bases de données, stockage, génération 3D, authentification, etc.) ?

Ça va ? Pas trop mal à la tête ? Nous n'aborderons pas dans ce dossier tous les services et fonctions backend qui nécessitent souvent d'intégrer des services externes (typiquement en mode Cloud) comme pour l'authentification (ex. : authentification Facebook), stockage via un stockage Web, un traitement d'analyse en mode Cloud, calcul et affichage d'une interface dynamique (ex. : jeux mobiles).

Après il faut raison garder. On ne va pas coder une "simple" app de gestion d'une entreprise avec les fonctions les plus avancées d'interface (ex. : WebGL, Unity3D) ou coder en C++ une simple app avec une base de données, ni utiliser SQL Server ou Oracle Database quand MySQL ou SQL Lite suffit largement. Bref, n'utilisez pas un tank quand un vélo suffira.

Des coûts à considérer

Tout développement a un coût, un budget. Par exemple, si une société Y fait développer une app mobile, elle doit considérer le salaire du développeur (qu'il soit interne ou externe). Et selon le profil du développeur, et des outils utilisés, la facture ne sera pas la même. Après, attention aussi : un profil performant a un coût généralement supérieur à la moyenne mais vous pouvez gagner en temps de développement…

Cet article détaille le coût du développement même si l'étude est centrée sur les USA ; cela peut donner une bonne idée : http://www.comentum.com/mobile-app-development-cost.html

Sur les salaires des développeurs mobiles, les profils Android et iOS ont toujours une majoration. Ainsi un profil débutant peut espérer +35 k€, voire, 40 k€ / an. Mais nous trouvons certaines fourchettes salariales trop extrêmes : 42-45 k€ en profil 0-2 ans expérience. La tendance actuelle est clairement d'avoir un profil multiplateforme et non plus sur une seule plateforme. •

Développement mobile : quelles solutions, comment choisir, quel outil pour quel développement ?

Arnaud Piroelle
@ArnaudPiroelle
Consultant Android,
Xebia

Florent Capon
Consultant iOS,
Xebia

Maxime Fontanier
Maxime.Fontanier@cnes.fr
Service DNO/DA/AQ
CNES

Sylvain Teodomante
Sylvain.Teodomante@cnes.fr
Service DNO/DA/AQ
CNES

Alors qu'il y a quelques années, nous pouvions encore nous poser la question, aujourd'hui c'est bien une réalité : le monde est mobile. Une donnée le prouve : en 2014, les ventes de terminaux mobiles ont dépassé celles des PC. Les utilisateurs naviguent davantage sur leurs smartphones et tablettes, et deviennent de plus en plus exigeants sur les performances, le temps de réactivité, l'ergonomie de l'interface, etc. Il devient indispensable d'intégrer cette évolution et de développer des applications de qualité.

Pour ce faire, il existe deux approches :

- Le web mobile : il s'agit de développer un site Web adapté au mobile, plus connu sous le nom de « responsive ». On parle également de « progressive WebApp » permettant d'optimiser le chargement des écrans mais aussi la gestion du mode offline.
- L'application mobile : l'application, spécifique au mobile, n'a pour cible que le smartphone, la tablette et la montre connectée. Son avantage est de proposer une interaction forte avec toutes les possibilités offertes par le hardware (appareil photo, gyroscope, etc.) et d'être présent dans les stores de chaque plateforme choisie (au contraire d'un site Web mobile).

Dans le cadre d'une approche Web Mobile, le choix de la technologie est large puisque identique au développement Web « traditionnel ». Il existe aujourd'hui de nombreux frameworks HTML / CSS / JS (dont notamment Angular, React.js ou Vue.js) pour développer un site mobile ; la plupart proposant des outils pour faciliter l'affichage responsive et les interactions avec le hardware. Ces réalisations ne représentant pas un « développement mobile » en tant que tel nous ne l'aborderons pas davantage au cours de cet article.

Dans la seconde approche, le choix de la technologie peut être plus délicat car les solutions possibles (native, hybride, transcompilée) présentent des orientations de développements très différentes avec des avantages et inconvénients marqués. Ce choix crucial impactera par la suite la performance, la maintenabilité, la pérennité, le coût, etc.

Développement

Trois solutions : native, hybride, transcompilée

Pour réaliser une application mobile qui a pour objectif d'être déployée sur un store (qu'il soit public ou d'entreprise), trois solutions techniques sont possibles :

- Les technologies natives : elles permettent de développer une application spécifique pour chaque plateforme (iOS, Android, Windows Phone), dans un langage supporté officiellement par l'éditeur de la plateforme.
- Les technologies hybrides : elles permettent d'encapsuler un site Web dans une application « coquille » native, afin de faciliter l'accès aux composants du dispositif à partir du code Web.
- Les technologies transcompilées : elles permettent de développer une seule application à l'aide d'un langage unique qui sera transformée en applications natives, pour chaque plateforme cible.

Quelle solution pour quel besoin ?

Choisir une solution technique pour le développement d'une application mobile n'est pas simple. De nombreux paramètres entrent en jeu, que ce soit le mode de distribution de votre application, les plateformes ciblées ou même les ressources disponibles pour le développement.

Dans le cas d'un déploiement multiplateforme, il n'existe pas de solution miracle, car chacune des technologies a ses forces et faiblesses. Le choix est plutôt une question d'équilibre, entre le temps consacré aux développements, les coûts engendrés, la pérennité de l'application, sa maintenance et la qualité souhaitée.

Pour des réalisations plus petites ou dites « jetables », l'hybride semble être une bonne option. Dans le cas inverse, pour une application plus complexe ayant pour vocation de durer dans le temps, le natif s'impose naturellement.

Pourquoi choisir le natif ?

Le développement natif consiste à développer une application pour chaque plateforme cible. Il est ainsi supporté officiellement par l'éditeur de la plateforme cible : Apple pour iOS, Google pour Android et Microsoft pour Windows Phone. Il est alors aisé de suivre les évolutions de chaque constructeur, OS et API proposés.

Un second avantage et non des moindres, ce sont les communautés de développeurs actives sur ces plateformes (surtout pour iOS et Android). Il est alors facile de trouver des réponses aux problèmes que l'on peut rencontrer. On trouvera aussi des milliers de librairies tierces pour tout type de besoin.

Le développement natif permet aussi de proposer la meilleure expérience utilisateur possible avec des performances optimales : il assure une pleine exploitation de toutes les capacités et ressources proposées par le device,

sans aucune couche intermédiaire venant réduire la mémoire ou les temps de calcul. Dans le cas d'applications à forte sollicitation du CPU ou GPU, la différence entre une application native ou hybride peut être très marquée : scrolling saccadé, freeze ou même crash si la mémoire n'est pas gérée correctement.

Autre grande force du natif : la gestion de l'affichage de listes de plusieurs centaines ou milliers d'éléments ; grâce à des composants dédiés avec un système de recyclage des cellules limitant la sollicitation CPU et mémoire, ce type d'affichage, devenu presque incontournable aujourd'hui, est facile à intégrer.

Nous parlions précédemment d'expérience utilisateur. La performance et la réactivité sont certes des points importants dans ce domaine, mais une autre composante est essentielle : l'ergonomie. En effet, chaque système d'exploitation et device dispose des codes ergonomiques qui lui sont propres. Ainsi, le respect de ces codes décrits par Apple, Google ou Microsoft au travers de documentations, est facilité par l'ensemble des outils natifs mis à la disposition du développeur. Leurs évolutions créent très peu d'impact en termes de temps de développement car sont anticipés par le constructeur.

Enfin, dernier point non négligeable, notamment dans la réalisation d'applications d'envergure visant à être pérennes : la maintenabilité, accrue grâce à une base de code saine écrite et développée spécifiquement pour une plateforme. Le code n'est alors pas pollué par des instructions qui pourraient être différentes d'une plateforme à une autre, contrairement à un développement hybride. Les risques de régression sont ainsi réduits puisqu'un correctif spécifique à une technologie n'aura aucun impact sur les autres plateformes. Les phases de recette et test s'en trouvent aussi allégées. [1]

Pourquoi choisir l'hybride ?

Un développement hybride implique une base de code unique qui sera la même d'une plateforme à une autre. Avantage considérable par rapport à un développement natif, cette base de code unique permettra un développement multiplateforme très rapide. Ceci permet de garantir l'ensemble des fonctionnalités et la maintenance de manière indépendante de la plateforme.

De plus, le développement hybride bénéficie d'un bon nombre de solutions techniques. Il y en a pour tous les goûts. Les technologies venant du monde du Web sont les plus utilisées. Depuis des dizaines d'années, les développeurs du monde entier utilisent HTML, CSS et JS afin d'offrir des sites Web de qualité aux utilisateurs. Rien d'étonnant à ce que ces technologies soient utilisées dans des solutions hybrides sur mobile. Représentant une très grande partie des développeurs sur le marché, il est plutôt aisé de trouver des profils qualifiés sur ces technologies pour réaliser un projet hybride.

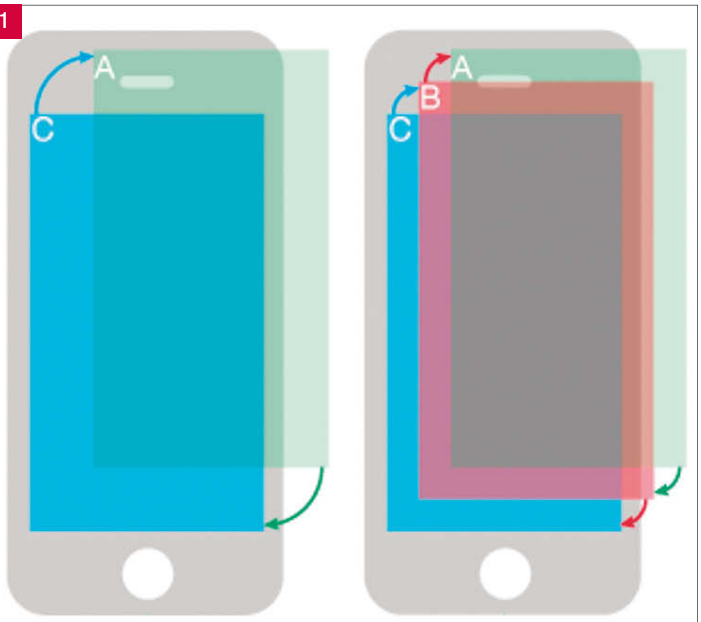
Cependant, ce type de développement a des limites. Que ce soit en termes de performances, de fonctionnalités liées aux hardwares ou de respect de l'expérience utilisateur, une solution hybride ne peut arriver au niveau d'un développement natif.

Pourquoi choisir le transcompilé ?

Au vu des avantages et inconvénients des 2 modes de développement proposés précédemment, de nouvelles solutions visent à garder le meilleur des deux mondes, à savoir une base de code unique avec les performances et les fonctionnalités des différentes plateformes. Parmi elles, on retrouve par exemple Xamarin ou encore AppCelerator.

Ces solutions permettent le développement d'une base de code unique qui sera transformée par la suite en code natif. Elles proposent un support au plus proche des plateformes, facilitant ainsi l'optimisation des performances et le respect de l'expérience utilisateur. Toutefois, ce support n'est pas magique, il faut garder en tête que chaque mise à jour de la plateforme implique une mise à jour du framework. Il faut donc un minimum de temps afin de bénéficier des nouveautés de chaque plateforme.

1



À gauche : développement natif ; à droite : développement hybride
A : La couche OS
B : La couche hybride (webview)
C : La couche physique (l'écran tactile)

Et des webviews dans du natif ?

Impossible de ne pas avoir entendu parler de « webview dans du natif ». Oui c'est possible, oui, ça se fait souvent. Néanmoins, il faut veiller à utiliser cette solution avec parcimonie car elle représente dans bien des cas une « roue de secours » ou un palliatif à un développement trop coûteux.

Même si cela peut présenter des avantages (délai, coût, etc.), les utilisateurs de votre application ne sont pas dupes et constateront un décalage ergonomique et souvent graphique entre vos parties développées en natif et les écrans affichés dans une webview. Si votre projet ressemble à un patchwork de technologies, vos utilisateurs risquent d'être déçus : la notation de votre application dans le store pourrait en pâtir.

Quel langage utiliser ?

Même s'il ne sera pas nécessairement un critère de choix, il peut-être important de prendre en compte quel langage est possible pour chaque solution technique.

Dans le cadre d'un développement natif, plusieurs solutions sont possibles selon les plateformes :

- iOS (Apple) : Swift remplace désormais le vieillissant Objective-C, même si ce dernier est toujours possible mais non recommandé. À noter que contrairement à l'Objective-C, Swift est open source et permet de développer des applications serveurs.
- Android (Google) : Java est le langage officiellement supporté par

Google mais d'autres sont possibles, comme Groovy ou même Scala… Néanmoins, ils ne sont que très anecdotiques car lourds à mettre en place et peu adaptés au développement mobile. Mais un nouveau langage a fait son apparition et peut changer la donne : Kotlin. Conçu par JetBrains en 2014, il est pensé comme un langage fonctionnel permettant de simplifier au maximum le code et de rendre la maintenance plus facile. Bien qu'assez jeune, Kotlin est parfaitement adapté au développement mobile sur Android.

- Windows Phone (Microsoft) : nativement, Microsoft propose trois langages différents. Chacun sera libre de choisir selon ses préférences :
  - C# / XAML : langage très proche de Java, le C# est très reconnu dans le monde du développement embarqué.
  - JS / HTML : JS et HTML sont très populaires dans le monde du Web. L'intégration de ces langages dans le développement mobile est un moyen d'attirer une nouvelle population de développeurs.
  - C++ / XAML : langage de programmation très populaire pour ses performances et son approche très bas niveau.

Dans le cas d'une solution hybride, le développeur pourra choisir le framework HTML / JS / CSS de son choix.

Pour les technologies transcompilées, le langage dépendra de la solution choisie. Par exemple, une application AppCelerator s'écrit en JavaScript alors qu'une application Xamarin se développe en C#.

Quelle stack technique choisir ?

Le monde de la mobilité évolue très vite et il n'est pas rare de voir une stack technique devenir obsolète en quelques mois seulement. Au risque donc d'être dépassés dans les prochains mois, nous avons souhaité vous présenter les stacks techniques recommandées à l'heure actuelle :

	iOS	Android	Windows
Client HTTP	URLSession ou Alamofire	OkHttp	Restsharp
Client API REST	URLSession ou Alamofire	Retrofit	Restsharp
Sérialisation / Désérialisation JSON	Unbox	Gson ou Moshi	Newtonsoft json
Gestion des images	AlamofireImage	Glide	Imaging SDK
Programmation réactive	RxSwift	RxJava	
Gestion des logs	CocoaLumberjack	Timber	Log4Net
Base de données locale	Realm	SQLBrite	SQLite
Injection de dépendances	Swinject	Dagger	MvvmLight
Crash reporting	Crashlytics	Crashlytics	HockeyApp
Tracking utilisateur	Fabric	Fabric	Application Insights

Ces stacks techniques doivent être maintenues à jour et modifiées régulièrement afin de suivre l'évolution de chaque plateforme.

Pour un développement hybride, c'est différent. Nous parlerons davantage de frameworks regroupant tous les outils nécessaires à la bonne réalisation d'une WebApp. Chacun ayant ses spécificités, patterns et architectures, le développeur pourra choisir celui avec lequel il se sent le plus à l'aise. Les plus connus aujourd'hui sont :

- AngularJS
- EmberJS
- React
- Vue.js

Quelle solution pour quel coût ?

Nous venons d'analyser, pour chaque technologie, les points forts et critères importants qui doivent aider à choisir une technologie plutôt qu'une autre. Néanmoins, la question du budget consacré au développement d'application peut contredire ce choix. Là où le natif paraît être la solution garantissant les meilleures performances et la maintenance la plus fiable, elle présente en revanche des coûts souvent plus élevés que ses concurrents. En effet, dans le cas d'une réalisation multiplateformes, le natif oblige l'équipe à développer une version de l'application pour chaque technologie supportée. Le coût de réalisation est alors multiplié par le nombre de plateformes souhaitées. Chaque technologie ayant son propre langage et ses codes ergonomiques, il sera difficile de capitaliser d'une plateforme à une autre. De même, les profils intégrés dans les équipes seront spécialisés sur une plateforme mobile alors que du côté hybride, il est possible pour un développeur full stack de travailler sur des projets desktop et mobiles.

Outre ceci, il faut aussi prendre en considération le coût des IDE, réel avantage budgétaire pour un développement natif. En effet, chaque constructeur propose gratuitement son propre IDE, alors que certaines solutions hybrides ou transcompilées sont payantes pour un usage commercial comme Xamarin ou AppCelerator.

Le coût des licences pour déployer son application sur les stores est quant à lui identique pour chaque type de développement. Pour distribuer une application dans un store (public ou privé), il est nécessaire de souscrire à une licence de développement qu'il s'agisse d'un développement natif, hybride ou transcompilé. À noter qu'Apple présente les tarifs les plus élevés : 99 / an pour l'App Store à 299 / an pour un store d'entreprise, contre 25$USD à vie pour Google (soit environ 24) et de 14 à 75 à vie pour Microsoft.

CONCLUSION

De nombreux critères entrent en jeu pour choisir la solution la plus adaptée à son besoin. Or, il n'existe pas de solution miracle. Tout est question d'équilibre entre les différents critères pour pondérer votre choix. Le plus simple reste de synthétiser tous ces éléments dans un tableau :

	Natif *iOS, Android, Windows Phone*	**Hybride** *Apache Cordova, PhoneGap, Ionic, etc.*	**Transcompilé** *Xamarin, AppCelerator, etc.*
Communauté	⊕	⊕	⊖
Compétences sur le marché	⊖	⊕	⊖
Performances	⊕	⊖	⊕
Expérience utilisateur	⊕	⊖	⊖
Capitalisation du code pour d'autres supports	⊖	⊕	⊖
Maintenabilité	⊕	⊖	⊖
Délai de réalisation pour un gros projet	⊕	⊖	⊖
Délai de réalisation pour un petit projet	⊖	⊕	⊕
Coût de réalisation	⊖	⊕	⊕

Projet mobile **Simple CRM**,

questions – réponses avec Brice Cornet

Quelle était la problématique de départ et les défis et problèmes d'une version mobile de la solution ?

La problématique de départ est juste que l'on était mauvais en mobile. Simple CRM avait pourtant pris le tournant du mobile très tôt. Dès qu'une connexion depuis un téléphone mobile a été possible, on a créé une interface « responsive », innovant à l'époque. Cette interface nous a permis de conquérir les cœurs les utilisateurs BlackBerry et des premiers smartphones HTC. Le souci c'est que l'on s'est un peu endormis sur nos acquis. On n'a pas vu venir la domination des apps mobiles. On avait, en 2015, sorti une application multiplateforme (iOS, Android, BlackBerry et Windows Phone), écrite en Java (voir PROGRAMMEZ, mai 2015, page 47), qui permettait de créer des sociétés, des contacts et des évènements dans Simple CRM, en mode déconnecté. C'était fonctionnel mais cela ne permettait pas de gagner des clients. En effet, à partir de 2015, on a vu arriver sur le marché des CRM des éditeurs US qui attaquaient le secteur via le mobile. On a fait l'erreur de ne pas y croire, de se dire que jamais une personne ne choisirait le logiciel qui est la colonne vertébrale de son entreprise, sur base d'un critère aussi superficiel que son ergonomie sur un écran de 5 pouces. On a eu à la fois raison et à la fois tord. Il est vrai que jamais une entreprise « sérieuse » ne posera réellement ce choix, car ses employés travaillent depuis de PC et ce qui l'intéresse c'est bel et bien la richesse de l'applicatif en termes de fonctionnalités. Par contre, il est vrai qu'un auto-entrepreneur peut poser ce choix. De plus, ces concurrents proposent des interfaces très design, qui ringardisaient totalement notre produit mobile. Le constat a été un peu difficile à accepter, puisque la réalité de l'analyse a été que l'on était trop vieux… La moyenne d'âge chez Simple CRM est de 38 ans et la plupart des membres de notre équipe ne sont pas très branchés mobiles. On en a une utilisation pratique, fonctionnelle, mais aucun de nous n'a de fibre sentimentale avec son joujou. Il fallait se rendre à l'évidence : on ne raisonnait pas avec la « passion mobile » qui touche une certaine génération. Du coup, la première étape a été de trouver du sang neuf, qui est nativement orienté mobile.

Comment les équipes ont-elles choisi les environnements et les outils de dév pour faire la version mobile ?

On a inversé la pyramide hiérarchique et on a laissé le sang neuf poser les choix au lieu de la direction technique. On a eu la chance de rencontrer un codeur formidable, qui s'appelle Nicolas. C'est un vrai passionné qui aime aller au bout des choses. On lui a donc expliqué comment la direction technique pose ses choix stratégiques :

- Etudes des solutions
- Test rapide des solutions
- Etude de l'adéquation entre les possibilités des solutions et le cahier des charges
- Durabilité de la solution

C'est là que l'on a vu la différence de génération car Nico a eu une toute autre approche. Lui a, en réalité, pris le problème le plus complexe, le fait de travailler en mode déconnecté, et est parti de cette contrainte pour poser son choix. Simple CRM a une structure DB en MySQL très costaude. Son point de vue était qu'il fallait déminer cela, pour ensuite trouver la bonne solution. Rapidement, il a choisi comme angle d'attaque le storage d'Angular.js. C'est sur cette base, qu'il a lancé son audit des solutions existantes. Il a repéré une adaptation de storage d'Angular.js réalisée par Ionic. De là, il a installé Ionic framework (http://ionic.io/) et ses tests ont confirmé que cette frame répondait à nos besoins.

Comment s'est déroulé le projet ? Et les problèmes rencontrés ?

Au niveau technique, la première étape a été de moderniser le cœur logiciel de Simple CRM.

C'est toujours un peu le soucis quand tu veux ouvrir une nouvelle porte fonctionnelle : tu te rends parfois compte qu'il y a pas mal de poussières derrières la porte… Il ne faut pas oublier que le code de Simple CRM évolue depuis 10 ans. Dès lors, la première étape a été de tuer notre système de webservice et de le remplacer par une API. Ensuite, il a fallut injecter dans notre stratégie commerciale un nouveau canal de vente : les appstores. Je plaide pour un meilleur dialogue entre les codeurs, la vente et le marketing ; l'union fait la force et personne n'a le monopole des bonnes idées. Ici, il a fallu clairement que les codeurs évangélisent les autres. Il a fallut définir une stratégie où des clients potentiels peuvent accéder à une version « freemium » de Simple CRM. Au niveau du projet de coding pure et dure, rien de différent d'un projet « classique » : cahier des charges, dev, tests, audit de sécurité, revue IHM, user test et découpage du projet en v1, v2 et v3. Côté soucis technique : rien à signaler. Les mobiles et tablettes sont des machines aux performances et stabilités éprouvées, idem pour le framework. Ionic revient à créer une appli web qui tourne tout simplement en local, et l'appli web. La seule différence c'est que l'on n'a pas de DB et que le duo CPU/RAM est plus modeste.

La solution choisie fut-elle finalement la bonne ?

Impossible pour moi d'y répondre de façon factuelle. Je dirai que vu la vitesse et le confort de développement qu'offre ce framework, je serai tenté de dire que oui. Après, coder le cœur de la bête, c'est 10% du travail. 90% du travail, c'est la maintenance et l'évolution de ce code. Donc la question serait plutôt : est-ce que Ionic tiendra sur la longueur ? Difficile de répondre. Par contre, on sait que les composants centraux comme Angular.js est un framework javaScript libre et open-source développé par Google. Donc on est assuré d'une certaine pérennité qui tend à nous rassurer sur la qualité de notre choix. •

Témoignage
L'AVIS DE GAËL

- Gaël (@gael_duval) - Créateur Mandrake-Linux 1998, Co-Fondateur NFactory.io en 2016

L'avantage principal que je vois au développement hybride est de maintenir une seule branche pour le code ce qui se traduit mécaniquement par une réduction du coût de développement (un seul dév est nécessaire) et une simplification ultérieure de la maintenance. Autre avantage, en particulier pour Ionic : l'utilisation de HTML5+CSS pour le développement de l'interface utilisateur est vraiment accessible à beaucoup de monde. Donc ça peut permettre à un non-expert de faire des petites modifications après coup en restant relativement serein. L'inconvénient ce sont des performances un peu moindres par rapport à du natif (réactivité de l'appli), mais avec du matériel moderne, une personne non avertie n'est souvent pas capable de voir la différence pour une application pas trop lourde et ne requérant pas trop de performances multimedia. Pour notre incubateur-accélérateur de startups NFactory.io, on recommande donc hybride (Ionic) quand le coût de dev et les délais de développement sont clé, et du natif en cas de besoin important de performance, ou quand l'appli est vraiment lourde (beaucoup, beaucoup de vues...).

Par expérience aujourd'hui on peut faire développer une application hybride (basique) pour moins de 10K en Asie. En Europe et en natif Android + iOS on tape rapidement au-dessus de 50K .

Je constate en ce moment que les boites de dev proposent de plus en plus React Native, et il me semble que NativeScript est très prometteur pour envisager du développement hybride avec des performances proches du natif.

Enfin, dans tous les cas, hybride ou pas, les modules tiers d'authentification (via Facebook ou Google+ par exemple), sont souvent des moments difficiles dans le développement et les phases de test. •

Pourquoi (choisir) Xamarin ?

J'ai eu à plusieurs reprises l'occasion de parler de Xamarin en conférence, en meetup et bien évidemment dans plusieurs numéros du magazine que vous tenez entre vos mains. L'objectif de cet article n'est pas de revenir sur les intérêts de Xamarin par rapport à d'autres plateformes de développement mobile, mais plutôt de vous parler de mon expérience et de ce qui m'a conduit à être un fervent défenseur de cette technologie aujourd'hui.

John Thiriet
Consultant Formateur chez **Cellenza**
Microsoft MVP Visual Studio and Development Technologies, Windows Development Xamarin MVP

Un peu d'historique

J'ai commencé le développement .NET il y a de cela une dizaine d'années, lors de mon cursus à Epitech. Mes premiers développements .NET ciblaient déjà du mobile puisque je devais faire un client Windows Mobile 5. Au fil des avancées de C# et de .NET, je me suis attaché à la plateforme, et j'ai eu l'occasion de travailler sur à peu près tous types de supports ; que ce soit le Web avec ASP.NET MVC, les Web services avec les ASMX ou WCF, les clients lourds avec WPF, les clients hybrides avec Silverlight, les applications tactiles multipoints sur les anciennes tables Microsoft Surface renommées PixelSense. Pendant longtemps, avec C# et .NET, je n'ai pas senti de limites particulières à ce que je pouvais faire.
Mais ça, c'était avant l'apparition de l'iPhone.

L'iPhone

Pour la première fois, je m'étais senti frustré de ne pas pouvoir développer sur un support qui m'intéressait beaucoup. J'ai tenté de regarder Objective-C mais la syntaxe me rebutait, et d'autres choses m'occupaient à ce moment-là. J'ai continué quelques temps en tant que simple utilisateur d'iPhone en me disant « un jour c'est sûr je ferai une application ».

Windows Phone 7

Motivé par la mobilité et enthousiasmé par la mise à disposition du SDK, j'ai fait mes premières armes en développement mobile tactile sur Windows Phone 7 en réutilisant toutes les compétences .NET et XAML que j'avais acquises sur Silverlight et WPF. L'évolution était donc pour moi très simple et logique. J'ai dès lors continué quelques années sur ce secteur avec WP8, Windows 8 et Windows RT, n'ayant que peu de travail à faire à chaque nouvelle version pour me mettre à jour.

Xamarin

Pendant tout ce temps je suivais de loin l'écosystème Mono et c'est avec grand intérêt que j'ai observé l'évolution de MonoDroid et MonoTouch en Xamarin. La coïncidence est que la véritable stabilisation de Xamarin natif arrivait au moment où les projets Windows Phone et Windows 8 se faisaient rares. Le moment était venu pour moi de tester Xamarin et de développer mes premières applications Android et iOS. Contrairement à toutes mes tentatives précédentes de changement de brique .NET où tout s'était très bien passé, avec Xamarin ce ne fut pas le cas. Cela peut paraître idiot mais iOS et Android ne sont pas des plateformes nativement .NET, et cela change tout !
Fini donc les transitions faciles et place au mode extrême.

La découverte des nouveaux mondes

J'ai troqué ma machine sous Windows pour un MacBook Pro. Je me suis familiarisé avec ce système que je n'avais jamais utilisé auparavant. J'ai installé Windows dans un Parallels et à ma grande surprise tout s'est merveilleusement bien passé. Les performances sont là et au final je me suis assez vite habitué à macOS.
Il m'a fallu monter en compétence sur deux plateformes en parallèle. Deux systèmes qui ont des fonctionnements très différents des applications de l'écosystème XAML. A ce moment-là, Xamarin Forms n'en était qu'à ses balbutiements et d'ailleurs j'ignorais son existence. Xamarin rimait donc toujours avec approche native. J'ai passé pas mal de temps sur Pluralsight à regarder des cours sur le développement Android en Java et iOS en Objective-C et plus tard en Swift. J'ai dû apprendre à lire du code iOS, à comprendre ce qu'est un protocole, à décrypter des API en Java et à tenter de maîtriser le cycle de vie des Fragments et des Activities. J'ai fait quelques POC, je me suis habitué à Xamarin Studio. J'ai utilisé Android Studio, joué avec le SDK Manager pour comprendre pourquoi rien ne marchait jamais et au final maudire les Google Play Services. J'ai appris à utiliser Xcode pour le design d'interfaces graphiques en étant dubitatif au premier abord avec le système de contraintes d'iOS. Plus tard j'ai pu profiter d'un accès à la Xamarin University et ainsi solidifier les connaissances que j'avais acquises en natif pour les adapter en Xamarin. Après de longs mois d'auto-formation, un monde complet et un peu hostile au développeur .NET que je suis, a donc fini par s'ouvrir à moi.

Les problèmes

On parle souvent des bugs dans Xamarin. Comme tout logiciel, Xamarin n'en est pas exempt. Mais finalement, dans son approche classique, Xamarin n'est qu'un wrapper aux APIs natives de la plateforme. La plupart du temps, les bugs et problèmes que je rencontre viennent de la plateforme sous-jacente. J'ai fini par réaliser que les meilleures documentations Android et iOS existantes étaient les fils de discussions sur StackOverfow. On y trouve des réponses en Java, en Objective-C et en Swift très faciles à transposer en C#.
En ce qui concerne Xamarin Forms c'est une autre histoire. Les bugs sont plus spécifiques à cette surcouche graphique et les meilleures réponses vont se trouver dans les forums de Xamarin. La stabilité de ce dernier laisse parfois à désirer et il ne faut pas se jeter sur les toutes dernières version Beta ou Alpha sous peine de rencontrer pas mal de soucis. Auparavant les performances n'étaient pas véritablement au rendez-vous sur Android. Les choses se sont bien améliorées depuis et ce qui pèche le plus à présent est la stabilité de la version UWP de Xamarin Forms. Mais tout cela s'améliore grandement depuis le rachat par Microsoft et je suis très optimiste pour la suite.

Mais finalement pourquoi Xamarin ?

Je reste attaché au langage C# et à .NET avec toutes les fonctionnalités telles que Linq, async/await, PCL et autres… C# 7 nous annonce de belles choses.
.NET n'a jamais fonctionné sur autant de plateformes. Je peux donc utiliser mes compétences un peu partout et ne pas me sentir restreint dans mes ambitions. Je trouve que Xamarin a fait un travail remarquable, permettant de conjuguer le meilleur de tous les mondes en produisant des applications natives et performantes tout en autorisant une très grande portabilité du code. En combinant cet écosystème avec Microsoft Azure, Xamarin Test Cloud, Visual Studio Team Services et une approche DevOps, on arrive maintenant à produire des applications professionnelles avec des pratiques de développement idéales. Mais bon tout ceci fera surement l'objet d'un autre article. •

Créez des applications multi-plateformes pour **Android** et **iOS** avec Flutter

• Sylvain Saurel
sylvain.saurel@gmail.com
Développeur Mobile
http://www.ssaurel.com

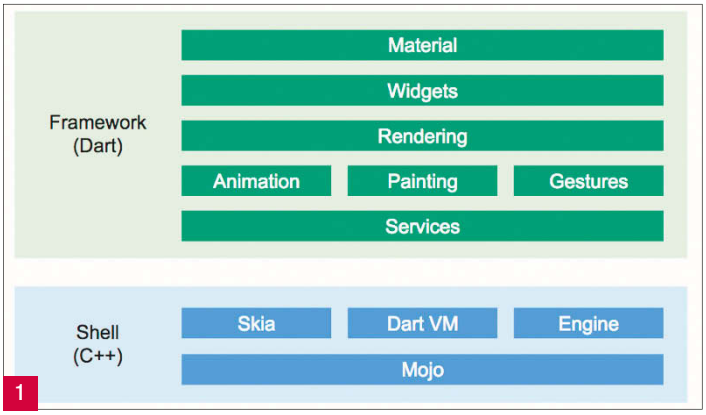
De par son omniprésence, le Mobile est désormais devenu un passage obligé pour les entreprises. Les développeurs d'applications mobiles ont le vent en poupe et désormais la question n'est plus de savoir si une société doit proposer une application mobile à ses utilisateurs, mais comment elle peut proposer, à moindres coûts, une application couvrant les 2 plateformes phares que sont Android et iOS. Pour tenter de répondre à cette problématique, Google propose Flutter, un framework open source qui n'en est qu'à ses débuts mais qui semble prometteur. Nous vous proposons de le découvrir dans cet article.

Développer des applications mobiles multi plateformes tournant sur Android et iOS constitue une tâche complexe et coûteuse lorsqu'il s'agit de tenir compte des spécificités de chacune des plateformes durant la conception et la réalisation. Ainsi, il est nécessaire d'avoir des ressources compétentes sur chacune de ces plateformes, ce qui représente un coût certain. Pour s'affranchir de ces coûts et proposer un support Mobile multi plateforme à partir d'une base de code unifiée, de nombreux frameworks voient le jour depuis plusieurs mois maintenant. Ils visent tous le même objectif : simplifier le développement d'applications mobiles multi plateformes. Cependant, les approches retenues pour tendre vers cet objectif ne sont pas les mêmes. Parmi ces frameworks, nous allons nous intéresser à un des derniers nés, à savoir Flutter.

Soutenu par Google, Flutter n'en est qu'à ses débuts mais l'approche adoptée par les équipes du géant de Mountain View est intéressante. Ainsi, Flutter vise à offrir aux développeurs une solution simple et efficace pour construire et déployer des applications mobiles multi plateformes, tout en offrant des performances de haut niveau à l'exécution sur les plateformes Android et iOS. En ce sens, Flutter poursuit le même graal que les autres frameworks mobiles multi plateformes déjà existants. Alors en quoi Flutter peut-il être considéré comme unique ? Flutter se différencie de ses concurrents car il ne s'appuie ni sur une WebView, ni sur les Widgets proposés par l'OS du périphérique d'exécution. Au lieu de ça, Flutter utilise son propre moteur de rendu graphique pour dessiner et afficher les Widgets sur l'OS cible. Les équipes en charge de Flutter travaillent d'ailleurs d'arrache-pied pour rendre ce moteur de rendu ultraperformant afin d'atteindre un niveau de FPS (Frame Per Second) équivalent à ce que proposent des applications natives. En ce sens, les solutions s'appuyant sur du HTML 5 embarqué dans une WebView ne peuvent lutter.

Le second point fort de Flutter réside dans l'expérience de développement qu'il propose aux développeurs. Celle-ci réside autour de 3 grands axes :

- Contrôle afin de donner aux développeurs l'accès à toutes les couches du système ;
- Performance afin de délivrer aux utilisateurs des applications fluides et responsives ;
- Fidèles dans le but de proposer aux utilisateurs un rendu fidèle et une expérience de même niveau que sur les plateformes Android et iOS.



1

Figure 1 : Architecture de Flutter

Architecture

D'un point de vue technique, Flutter est construit autour de 2 grandes parties détaillées dans le diagramme de la figure 1.

La partie framework utilise ainsi Dart, un langage maison poussé par Google depuis quelques années maintenant. Elle contient notamment les différents Widgets proposés aux développeurs. La partie Shell de Flutter embarque une machine virtuelle Dart et est construite en C/C++ avec l'utilisation du moteur de rendu 2D Skia et de la technologie de communication inter processus Mojo IPC. Pour exécuter le même code écrit en Dart sur Android et iOS, Flutter adopte 2 fonctionnements différents. Sur Android, le code C/C++ du moteur est compilé avec Android NDK et la grande majorité du code du framework et de l'application s'exécute en tant que code natif compilé par le compilateur Dart. Sur iOS, le code C/C++ du moteur est compilé avec LLVM, et le code Dart est compilé en mode AOT (Ahead Of Time) vers du code natif. L'application s'exécute de fait en utilisant un ensemble d'instructions natives.

Installation

Les fondements théoriques de Flutter présentés, nous pouvons désormais passer à son installation. Il est bon de noter qu'à l'heure actuelle, Flutter ne supporte que les systèmes d'exploitation Mac et Linux 64-bits. Un support Windows est cependant planifié dans le futur. La première étape va donc consister à récupérer le SDK de Flutter sur votre poste de travail depuis son dépôt GitHub via la ligne de commande suivante au sein d'un terminal : git clone https://github.com/flutter/flutter.git

Une fois le SDK téléchargé, il sera intéressant d'ajouter l'exécutable flutter à votre path comme suit :

```
export PATH=`pwd`/flutter/bin:$PATH
```

Afin de vérifier si votre système d'exploitation possède déjà toutes les dépendances nécessaires au bon fonctionnement de Flutter et le cas échéant de télécharger automatiquement celles venant à manquer, il faudra taper la commande suivante :

```
flutter doctor
```

Flutter se chargera alors de télécharger pour vous les dépendances absentes (figure 2) ce qui peut prendre un certain temps puisque cela concerne la plateforme Android mais également la plateforme iOS.

Configuration

Pour développer vos applications sous Flutter, plusieurs solutions s'offrent à vous. La plus basique étant d'utiliser un simple éditeur de texte. Bien entendu, cette solution est relativement limitée. Pour aller plus loin, il est possible d'utiliser l'éditeur Atom qui bénéficie d'un plugin Flutter dédié accessible ici : https://atom.io/packages/flutter . De notre côté, nous avons choisi la solution la plus intégrée avec le recours à l'éditeur IntelliJ IDEA gratuit dans sa version Community et qui est téléchargeable à l'adresse suivante : https://www.jetbrains.com/idea/download/ . Une fois IntelliJ IDEA installé, il restera à télécharger les plugins spécifiques Dart et Flutter afin d'avoir une expérience de développement la plus poussée possible. Pour les développeurs Android habitués à utiliser Android Studio, il faut noter que ces plugins ne fonctionnent pas sur ce dernier pour le moment. Le téléchargement des plugins se fait en allant dans le menu Préférences > Plugins, puis en effectuant une recherche au sein des dépôts de plugins (figure 3).

```
MacBook-Pro-de-Sylvain:Flutter ssaurel$ flutter doctor
Downloading Dart SDK 1.21.0-dev.9.0...
######################################################################## 100,0%
Building flutter tool...

 ╔════════════════════════════════════════════════════════════════════╗
 ║                Welcome to Flutter! - https://flutter.io            ║
 ║                                                                    ║
 ║ The Flutter tool anonymously reports feature usage statistics and basic ║
 ║ crash reports to Google in order to help Google contribute improvements to ║
 ║ Flutter over time. See Google's privacy policy:                    ║
 ║ https://www.google.com/intl/en/policies/privacy/                   ║
 ║                                                                    ║
 ║ Use "flutter config --no-analytics" to disable analytics reporting. ║
 ╚════════════════════════════════════════════════════════════════════╝

Downloading Material fonts...                       4867ms
Downloading package sky_engine...                   2621ms
Running 'flutter packages get' in sky_engine...     295ms
Building Dart SDK summary...                        1215ms
Downloading engine artifacts android-arm...         42137ms
Downloading engine artifacts android-arm-profile... 36493ms
Downloading engine artifacts android-arm-release... 30720ms
Downloading engine artifacts android-x64...         22784ms
Downloading engine artifacts android-x86...         22613ms
Downloading engine artifacts ios...                 47356ms
Downloading engine artifacts ios-profile...         46739ms
Downloading engine artifacts ios-release...         35550ms
Downloading darwin-x64 tools...                     62686ms
Downloading android-arm-profile/darwin-x64 tools... 7434ms
Downloading android-arm-release/darwin-x64 tools... 6425ms
[✓] Flutter (on Mac OS, channel master)
    • Flutter at /Users/ssaurel/Downloads/Flutter/flutter
    • Framework revision 1d2f7e3c02 (3 hours ago), 2016-11-28 21:47:40
    • Engine revision a8b7631b7a
    • Tools Dart version 1.21.0-dev.9.0

[✓] Android toolchain - develop for Android devices (Android SDK 23.0.2)
    • Android SDK at /Users/ssaurel/Library/Android/sdk
    • Platform android-23, build-tools 23.0.2
    • Java(TM) SE Runtime Environment (build 1.8.0_74-b02)

[-] iOS toolchain - develop for iOS devices (Xcode 8.1)
    • XCode at /Applications/Xcode.app/Contents/Developer
    • Xcode 8.1, Build version 8B62
    x ideviceinstaller not available; this is used to discover connected iOS devices.
      Install via 'brew install ideviceinstaller'.
    x ios-deploy not available; this is used to deploy to connected iOS devices.
      Install via 'brew install ios-deploy'.

[x] Flutter IDE Support (No supported IDEs installed)
    • IntelliJ - https://www.jetbrains.com/idea/

[✓] Connected devices
    • SM G930F • 988673343643493545 • android-arm • Android 6.0.1 (API 23)
```

2

Figure 2 : Installation des dépendances de Flutter

Pour terminer la configuration d'IntelliJ IDEA, il vous restera à renseigner le chemin vers le SDK Flutter à partir des Préférences et du menu "Langages et Frameworks" (figure 4).

Première application Flutter

L'installation du SDK Flutter et de l'environnement de développement associé terminée, il est temps de réaliser une première application Flutter. Pour ce faire, nous créons un nouveau projet Flutter sous IntelliJ IDEA. Par défaut, un premier projet basique est créé permettant de découvrir l'arborescence type d'un projet Flutter (figure 5).

On retrouve ainsi les répertoires Android et iOS contenant les fichiers nécessaires au fonctionnement de l'application sur ces 2 plateformes. Le répertoire lib contient le code Dart de l'application développée. Le fichier pubspec.yaml permet de définir les dépendances nécessaires au projet. Ici, seule la présence du SDK Flutter est nécessaire :

```
name: MyFlutterProject
description: My First Flutter Project.
dependencies:
  flutter:
    sdk: flutter
```

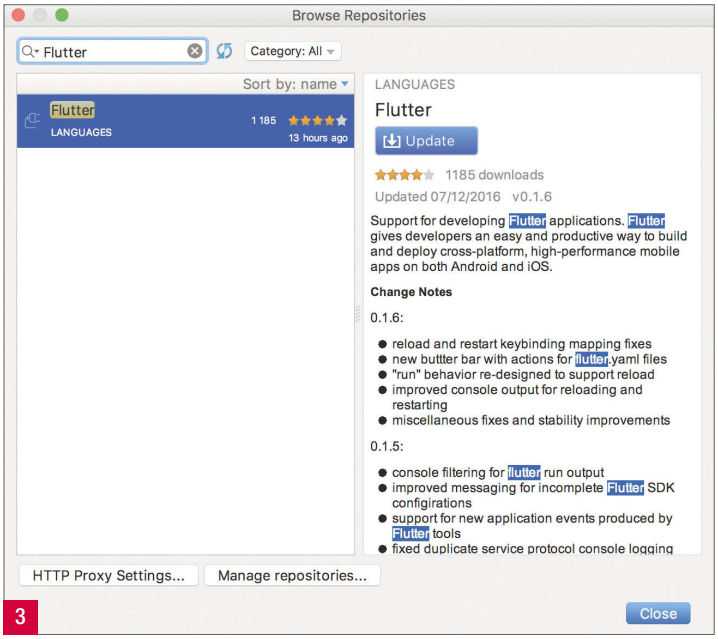


Figure 3 : Installation des plugins Dart et Flutter

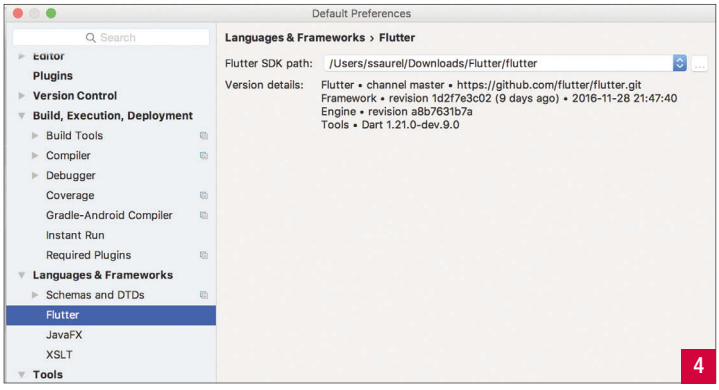


Figure 4 : Définition du Path du SDK Flutter

Enfin, le fichier flutter.yaml permet de définir un certain nombre d'options applicables au projet. Dans ce premier projet, on peut ainsi noter le positionnement à true de la propriété uses-material-design afin de préciser que l'application utilisera un rendu Material Design et donc la bibliothèque de Widgets Dart correspondante.
Au niveau du code de l'application, le point d'entrée est une méthode main() au sein de laquelle on appelle la méthode standard runapp() avec en entrée la classe représentant l'application :

```
import 'package:flutter/material.dart';

void main() {
 runApp(new MyApp());
}

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return new MaterialApp(
   title: 'Flutter Demo',
   theme: new ThemeData(
    primarySwatch: Colors.blue,
   ),
   home: new MyHomePage(title: 'Flutter Demo Home Page'),
  );
 }
}
```

La méthode build renvoie le Widget racine de l'application. Ici, on choisit donc de s'appuyer sur la classe MaterialApp à laquelle on précise le titre de l'application. On définit également le thème utilisé et enfin la page à afficher au lancement de l'application. On remarque ici que notre classe MyApp étend la classe StatelessWidget ce qui implique qu'on aura à faire à un Widget sans état. A contrario, la page d'accueil définie via la classe MyHomePage étend la classe StatefulWidget. Ceci implique que le Widget est avec état et doit donc définir un objet pour gérer son état renvoyé via la méthode createState() :

```
class MyHomePage extends StatefulWidget {
 MyHomePage({Key key, this.title}) : super(key: key);

 final String title;

 @override
 _MyHomePageState createState() => new _MyHomePageState();
}
```

La classe _MyHomePageState étend l'objet State et va nous permettre de définir le contenu de notre interface graphique. On crée ici au sein de la méthode build un objet Scaffold simplifiant la création de l'interface. Le framework Flutter étant optimisé, ses créateurs conseillent de recréer à chaque appel le contenu de l'interface plutôt que d'effectuer des mises à jour plus contraignantes et fastidieuses. Au sein de l'objet Scaffold, on définit le contenu de la barre d'action via la classe AppBar, le corps de l'interface, et, enfin, un bouton d'action flottant. Au niveau du corps de l'application, on définit simplement un composant texte qui affiche le nombre de clics reçus par le bouton d'action flottant. Enfin, au niveau du bouton d'action flottant, on définit une méthode callback en charge d'incrémenter la valeur du compteur de clics. Tout ceci nous donne le code suivant pour la classe _MyHomePageState :

```
class _MyHomePageState extends State<MyHomePage> {
 int _counter = 0;

 void _incrementCounter() {
  setState(() {
   _counter++;
  });
 }

 @override
 Widget build(BuildContext context) {
  return new Scaffold(
   appBar: new AppBar(
    title: new Text(config.title),
   ),
   body: new Center(
    child: new Text(
     'Button tapped $_counter time${ _counter == 1 ? '' : 's' }.',
    ),
   ),
   floatingActionButton: new FloatingActionButton(
    onPressed: _incrementCounter,
    tooltip: 'Increment',
    child: new Icon(Icons.add),
   ),
  );
 }
}
```

Pour exécuter l'application, il suffit de choisir sa plateforme d'exécution puis de cliquer sur le bouton run sous IntelliJ IDEA. Cela nous donne le résultat présenté à la figure 6 sous Android.

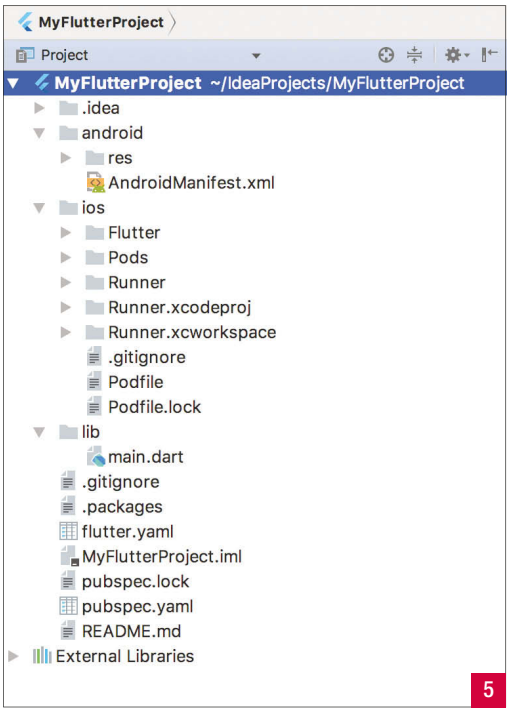


Figure 5 : Arborescence d'un projet Flutter

Une liste de livres ...

Pour aller plus loin dans notre découverte de Flutter, nous allons complexifier légèrement notre application en lui donnant comme objectif de télécharger une liste de livres ayant pour sujet le langage Dart. Au sein de notre fichier main.dart, nous définissons une liste de livres basée sur la classe Book qui propose un nom, une URL vers une image pour la couverture et enfin un auteur :

```
class Book {
  const Book({this.name, this.cover, this.author});
  final String name;
  final String cover;
  final String author;
}

final List<Book> _books = <Book>[
  new Book(name: 'The Dart Programming Language',
          cover: 'https://www.dartlang.org/assets/covers/dart-programming-language.jpg',
          author: 'Gilad Bracha'),
  new Book(name: 'Learning Dart - Second Edition',
          cover: 'https://www.dartlang.org/assets/covers/learning-dart.png',
          author: 'Ivo Balbaert and Dzenan Ridjanovic'),
  new Book(name: 'Dart By Example',
          cover: 'https://www.dartlang.org/assets/covers/dart-by-example.png',
          author: 'Davy Mitchell'),
  new Book(name: 'Dart Essentials',
         cover: 'https://www.dartlang.org/assets/covers/dart-essentials-sikora.png',
          author: 'Martin Sikora'),
];
```

Ensuite, nous devons réaliser 3 Widgets : le premier pour afficher un livre au sein de la liste et qui sera nommé BookListItem, le second pour rendre la liste entière à l'écran et enfin un callback permettant de gérer le click sur un élément de la liste :

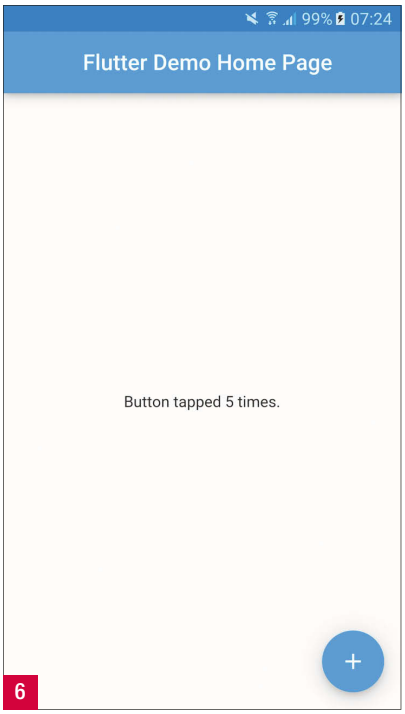


Figure 6 : Notre application Flutter sous Android

```
typedef void BookChangedCallback(Book book, bool read);

class BookListItem extends StatelessWidget {
  BookListItem({Book book, this.read, this.onBookChanged})
      : book = book,
        super(key: new ObjectKey(book));

  final Book book;
  final bool read;
  final BookChangedCallback onBookChanged;

  TextStyle _getTextStyle(BuildContext context) {
    if (!read) return null;

    return new TextStyle(
      color: Colors.black54,
      decoration: TextDecoration.lineThrough,
    );
  }

  @override
  Widget build(BuildContext context) {
    return new ListItem(
      onTap: () {
        onBookChanged(book, !read);
      },
      title: new Text(book.name, style: _getTextStyle(context)),
      leading: new Image.network(book.cover, fit: ImageFit.cover),
      subtitle: new Text('Written by ' + book.author, style: _getTextStyle(context)),
    );
  }
}

class FlutterBookList extends StatefulWidget {
  FlutterBookList({Key key, this.books}) : super(key: key);

  final List<Book> books;

  @override
  _FlutterBookListState createState() => new _FlutterBookListState();
}
```

Au sein de la classe BookListItem, on pourra remarquer que l'on utilise le réseau pour récupérer les images de couverture des livres via la méthode network de la classe standard Image. On notera également que les livres déjà lus (avec la propriété read à true) sont décorés avec un TextStyle particulier permettant de barrer le titre d'un livre déjà lu. Lors de la création de l'objet ListItem au sein de la méthode build, on renseigne le paramètre onTap afin d'appeler le callback onBookChanged lorsqu'un utilisateur va cliquer sur un livre indiquant par la même qu'il vient de le lire. La classe FlutterBookList est avec état car elle doit permettre de gérer la liste de livres affichée à l'écran. La gestion de l'état se fait au sein d'une classe _FlutterBookListState qui étend la classe standard State. Cette classe nous permet également de définir le contenu de l'interface graphique de notre application via l'emploi de la classe Scaffold. Nous définissons comme précédemment une barre d'action dans un premier temps. Ensuite, le corps de l'interface graphique sera composé d'une liste de

type MaterialList dont les enfants de type BookListItem sont construits à partir de la liste de livres définie au début du fichier main.dart . Cela nous donne donc le code suivant :

```
class _FlutterBookListState extends State<FlutterBookList> {
 Set<Book> _books = new Set<Book>();

 void _handleBookChanged(Book book, bool read) {
  setState(() {
   if (read)
    _books.add(book);
   else
    _books.remove(book);
  });
 }

 @override
 Widget build(BuildContext context) {
  return new Scaffold(
   appBar: new AppBar(
    title: new Text('Book List'),
   ),
   body: new MaterialList(
    type: MaterialListType.oneLineWithAvatar,
    children: config.books.map((Book book) {
     return new BookListItem(
      book: book,
      read: _books.contains(book),
      onBookChanged: _handleBookChanged,
     );
    }),
   ),
  );
 }
}
```

Avant de pouvoir lancer notre application, il nous reste à définir la classe FlutterBookList comme écran d'accueil pour notre application Flutter comme suit :

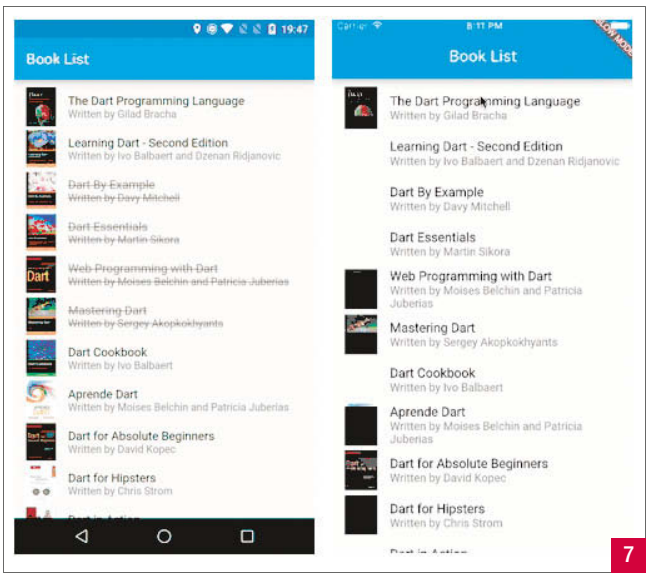


Figure 7 : Exécution sur Android et iOS

```
void main() {
 runApp(
  new MaterialApp(
   title: 'Flutter Book List',
   theme: new ThemeData(
    primarySwatch: Colors.blue
   ),
   home: new FlutterBookList(books: _books)
  )
 );
}
```

L'exécution sur la plateforme Android et sur la plateforme iOS nous permet de constater que nous obtenons ainsi 2 applications parfaitement fonctionnelles au design similaire avec un seul et même code comme base (figure 7).

Pour aller plus loin

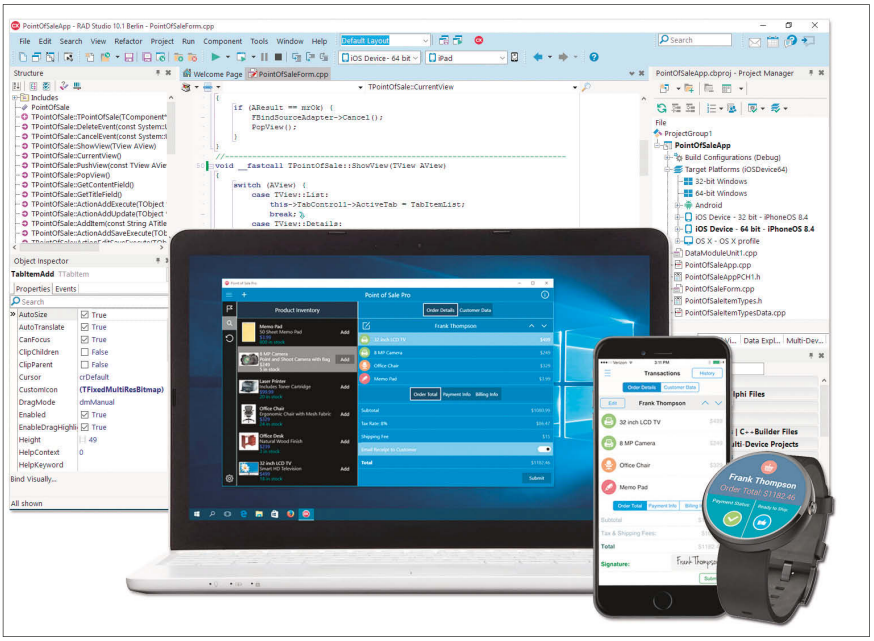
Encore jeune, le framework Flutter présente l'avantage de posséder d'ores et déjà un grand nombre de Widgets en standard. Il est ainsi aisé de réaliser des applications Material Design au rendu visuel soigné. Pour découvrir tous les Widgets à disposition, le site officiel de Flutter est une véritable mine d'or qu'il est important de consulter : https://flutter.io/widgets/ .
Bien évidemment, il n'est pas recommandé pour le moment de mettre en production des applications mobiles construites avec Flutter puisque certains bugs sont encore présents et doivent être fixés. Ainsi, durant la réalisation de notre application affichant une liste de livres dont les couvertures sont téléchargées via le réseau, nous avons pu constater certains problèmes pour récupérer les images sur iOS.
Ceci vient d'une implémentation de la partie réseau encore perfectible et comportant certains bugs. Gageons toutefois que les équipes de Flutter sauront corriger ces problèmes lors de la montée en puissance du framework dans les mois à venir.

CONCLUSION

Cette introduction à Flutter nous aura permis de découvrir un framework prometteur offrant une approche singulière vis-à-vis de ses concurrents pour répondre à la problématique du développement d'applications multi plateformes à partir d'une seule et unique base de code. En effet, Flutter se démarque d'une solution comme React Native par sa volonté de proposer son propre moteur de rendu visuel ouvrant ainsi les portes au fameux slogan "Write Once, Run Anywhere" pour les développeurs d'applications mobiles.
Ce choix d'architecture ne se fait pas sans inconvénients puisqu'au final cela implique qu'une application aura le même rendu visuel sur Android et iOS. Les utilisateurs Android, habitués au Material Design, y trouveront leur compte, là où les utilisateurs iOS pourraient être quelque peu désarçonnés du fait d'un design ne respectant pas les guidelines d'Apple.
Enfin, le choix du langage Dart par Google pour Flutter n'est certainement pas anodin.
Outre le fait que la plateforme Dart soit reconnue pour ses performances, il s'agit pour le géant de Mountain View de se préparer une éventuelle porte de sortie pour proposer une solution de développement d'applications mobiles s'affranchissant du langage Java et de la menace juridique toujours présente d'Oracle.

Delphi et le développement mobile

Questions – réponses avec Florian LOUIS, chef de projets R&D chez Ginkoia



Pouvez-vous présenter en quelques mots vos développements et votre activité mobile ?

Nous développons des solutions de gestion magasin, dans le domaine du sport, location, cycle, mode, chaussure et maroquinerie. Nos applications mobiles sont des extensions de notre logiciel principal, permettant à nos clients d'effectuer des tâches spécifiques directement en surface de vente ou de location, depuis un terminal de type smartphone.

Vous êtes plutôt développement natif, multiplateforme ou hybride ?

Dans notre activité, nous avons choisi de développer nos applications sur un langage multiplateforme. Nous utilisons le même langage pour nos applications Windows et pour nos applications mobiles.

Comment êtes-vous arrivé à utiliser Delphi pour faire des développements mobiles ?

Depuis sa création, notre logiciel principal est réalisé en Delphi. Nous avons fait évoluer nos développements en migrant nos projets au fur et à mesure que les versions de Delphi étaient publiées.

Lorsque nous avons envisagé de produire des applications mobiles pour nos clients, c'est assez naturellement que nous avons essayé de le faire sous Delphi. La version XE6 avec FMX nous proposait de compiler assez simplement des applications pour Android et pour iOS. Aujourd'hui nous développons nos applications sous Delphi 10 Seattle, et nous évoluons sur la dernière version (Delphi 10.1 Berlin). De plus, le fait de développer en Delphi nous a permis de faire entrer directement nos développeurs dans le projet sans formation lourde au préalable.

Avez-vous utilisé d'autres environnements de développement ?

A titre personnel, j'ai développé des applications essentiellement en natif (XCode pour iOS, Android Studio pour Android et Tizen), ce qui m'a permis de voir les avantages et les faiblesses du développement en code source natif.

Quels sont pour vous les atouts de Delphi dans le dév mobile ?

Un des atouts majeurs est de pouvoir compiler un programme aussi bien pour Windows (Win32 et Win64), iOS et Android avec très peu d'adaptation de code, tout au plus quelques directives de compilation conditionnelles en fonction de la plateforme cible. Une fois compilée, l'application générée est une application native pour chaque plateforme. Ceci nous permet de pouvoir développer nos applications sans forcément avoir l'appareil mobile avec nous. L'application fonctionne à l'identique en mode Win32 et nous permet de développer et tester nos fonctionnalités, avant de la compiler pour Android ou iOS afin de la tester sur l'appareil mobile. Un des autres avantages de Delphi par rapport à un développement dans le langage natif est de ne pas avoir à écrire plusieurs fois la même application si on destine celle-ci à être déployée sur plusieurs plateformes, ce qui est confortable. Le temps de développement et de mise au point est diminué d'autant. Delphi met aussi à disposition les outils pour réaliser simplement un serveur d'application (DataSnap), qui nous permet de développer assez facilement toute la couche d'échange de données entre le mobile et un serveur de données. Nos applications mobiles permettent essentiellement de consulter ou d'interagir avec des données stockées en base de données. Le trio FMX, Datasnap et FireDAC pour la connexion à différents types de moteurs de base de données fonctionne assez bien. Delphi nous permet d'obtenir un résultat exploitable avec un temps de développement assez réduit.

Comment jugez-vous la prise en charge des nouvelles API et des versions des systèmes mobiles dans Delphi ?

Delphi est un environnement en constante évolution. Nous avons régulièrement des mises à jour nous permettant de profiter de nouveaux composants. Pour Android, la mise à jour du SDK se fait avec les outils natifs. Les nouvelles API sont donc disponibles assez rapidement, et si l'une d'entre elles venait à manquer, il existe des outils pour générer les éléments nécessaires depuis les fichiers Java. D'origine, le langage Pascal permet de faire des inclusions en assembleur. Dans le même ordre d'idée, Delphi permet de faire des inclusions de portions de code accédant à des éléments en Java via JNI.

Quels outils supplémentaires utilisez-vous ?

Pour la compilation en iOS, nous devons utiliser un environnement sous macOS, avec XCode. Delphi fournit un utilitaire à installer pour pouvoir générer l'archive de l'application mobile sur Mac et pouvoir déployer nos applications en test sur des machines physiques (iPhone ou iPad). Pour la compilation Android, le système est plus simple. Delphi nous permet de déployer directement sur l'appareil mobile.

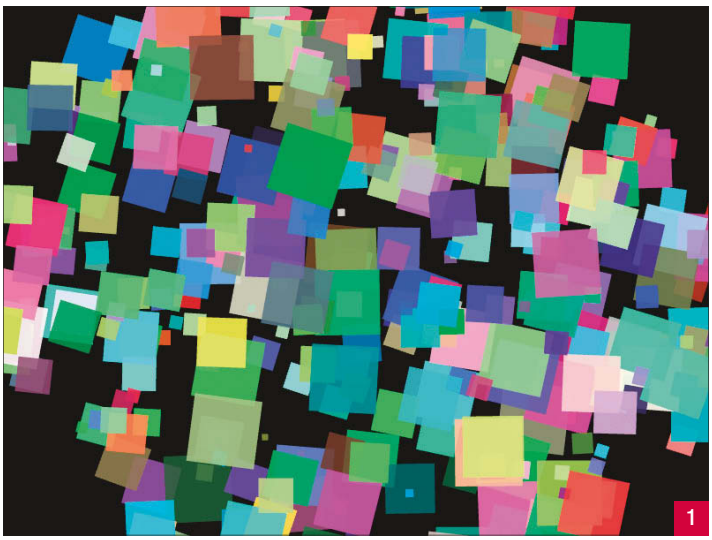
Pour rappel, Delphi est distribué par Barnsten qui est aussi le représentant d'Embarcadero en France. Site web : www.barnsten.com •

Exploser des pixels avec WebGL et JavaScript

Partie 2

• Denis Duplan
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Cet article est le second (et donc dernier) d'une série portant sur la production d'une animation de « pixels » s'éloignant d'un point d'origine en tournoyant (autour de leurs centres et autour du centre de l'écran), en grossissant et devenant toujours plus transparents jusqu'à disparaître de l'écran. [1]


1

Dans le premier article, nous avons vu comment créer le vertex shader (VS) et fragment shader (FS) et les alimenter en données pour que tous les « pixels » soient transformés et rendus en appelant une seule fois drawElements () par étape de l'animation.
Reste à voir comment produire l'animation. A étape régulière, il faut :

- Mettre à jour l'angle de rotation , le facteur de zoom Z et les composantes du vecteur de translation de chaque « pixel » ;
- Alimenter les shaders avec les données des points auxquelles ces données relatives à la transformation sont annexées ;
- Commander la transformation et le rendu des « pixels ».

La solution

β, Z, dX et dY constituent l'état d'un « pixel ». Il convient donc de créer un tableau contenant autant d'états qu'il y a de « pixels » et de modifier l'état de chaque pixel en incrémentant ces valeurs à chaque étape pour produire l'animation. Pour que chaque « pixel » semble avoir sa propre cinétique, la meilleure solution est de déterminer les incréments en les tirant au hasard dans des intervalles raisonnables, au sens où leurs bornes doivent être fixées empiriquement. L'état d'un « pixel » comprend donc ses valeurs de β, Z, dX et dY mais aussi leurs incréments spécifiques au « pixel ». Une fois calculées pour l'étape courante, les valeurs de , Z, dX et dY doivent être mises à jour dans le buffer associé à ARRAY_BUFFER afin qu'elles puissent être transmises au FS via l'attribut A_bzdxdy, comme précisé dans le précédent article. C'est le point faible de la stratégie consistant à limiter les appels à drawElements () : le volume des données par point est accru – en plus de ses coordonnées et de sa couleur, il faut y joindre l'état de son « pixel » -, et ces données doivent être transférées en mémoire à chaque étape.
Comme les données des points (X, Y, R, G, B) transmises via A_xy et A_rgb sont fixées une fois pour toutes, et que seuls les états des « pixels » (β, Z, dX, dY) transmis via A_bzdxdy sont modifiés à chaque étape, il est possible de limiter l'inconvénient en se contentant de mettre à jour ces états dans le buffer. WebGL dispose d'une fonction pour cela : bufferSubData (). Il s'agit donc d'appeler bufferData () une première fois pour associer à ARRAY_BUFFER un buffer de la taille requise (données des points et des états de leurs « pixels » respectifs) puis d'appeler bufferSubDate () pour mettre à jour le buffer partiellement.
Pour produire l'animation, la solution qui vient spontanément à l'esprit est de recourir à une des fonctions de l'objet window : setTimeout () qui serait rappelée à chaque étape, ou setInterval () qui serait appelée une fois pour toutes. Toutefois, ces fonctions sont bien trop imprécises pour assurer un rendu fluide – « dans la trame » pour reprendre un vocabulaire « old school » évoquant le délai entre deux balayages d'un même point par le faisceau d'électrons d'un tube cathodique à 60Hz. Fort heureusement, l'objet window dispose d'une autre fonction bien plus précise : requestAnimationFrame ().

Le code JavaScript

Comme précisé dans le premier article, le programme est structuré en trois parties dont il reste deux parties à explorer :

- L'initialisation de l'explosion assurée par explode () ;
- Le rendu d'une étape de l'explosion assurée par nextStep ().

Concernant explode (), le code est le suivant :
voir le code complet sur le site www.programmez.com

La logique

L'initialisation de l'explosion

Faisant le joint entre l'initialisation générale et le rendu d'une étape, l'initialisation de l'explosion concentre les opérations qu'il est impossible d'effectuer dans la première et dont il vaut mieux faire l'économie dans le seconde de ces phases. Choix est fait d'exposer une interface qui per-

Nombre de "pixels" :	400			
Nombre d'étapes :	200			
Dimensions d'un "pixel" :	0.01	par	0.01	
Vitesse de rotation générale (degrés) :	5			
Vitesse de rotation d'un "pixel" (degrés) :	Entre	5	et	20
Vitesse de zoom d'un "pixel" :	Entre	10	et	200
Vitesse de déplacement horizontal d'un "pixel" :	Entre	-10	et	10
Vitesse de déplacement vertical d'un "pixel" :	Entre	-10	et	10

2

met à l'utilisateur de modifier des paramètres de l'explosion pour déterminer de manière empirique les valeurs produisant l'effet recherché : [2]. Il s'agit donc de lire les valeurs de ces paramètres et de préparer une nouvelle animation en conséquence : créer les buffers de points et d'indices, modifier les valeurs des uniformes. Un « pixel » est décrit sous la forme de deux TRIANGLES. Créer les « pixels », c'est simplement créer les TRIANGLES qui les composent en boucle en alimentant deux tableaux : vertices[] et indices[]. Le nombre de pixels est nbPixels, les dimensions d'un « pixel » sont pixelWidth et pixelHeight :

```
var pVertices = [
  -pixelWidth / 2.0, pixelHeight / 2.0,
  pixelWidth / 2.0, pixelHeight / 2.0,
  pixelWidth / 2.0, -pixelHeight / 2.0,
  -pixelWidth / 2.0, -pixelHeight / 2.0,
];
var pIndices = [
  0, 3, 1,
  1, 3, 2
];

// Créer les "pixels"

vertices = new Array ();
indices = new Array ();
offsetI = 0;
for (i = 0; i != nbPixels; i ++) {
  r = Math.random ();
  g = Math.random ();
  b = Math.random ();
  j = 0;
  do {
    vertices.push (pVertices[j ++]);
    vertices.push (pVertices[j ++]);
    vertices.push (r);
    vertices.push (g);
    vertices.push (b);
  } while (j != pVertices.length);
  for (j = 0; j != pIndices.length; j ++)
    indices.push (pIndices[j] + offsetI);
  offsetI += 4;
}
```

A chaque « pixel » ses paramètres, regroupés dans une structure comprenant sa vitesse de rotation betaSpeed, sa vitesse de zoom zoomSpeed, ses vitesses de translations horizontale et verticale xSpeed et ySpeed. Chaque « pixel » dispose aussi d'un état, qui doit être reproduit pour chaque point à la fin du buffer des points. Ici encore, paramètres et états sont créés en boucle :

```
alpha = 1.0;
alphaSpeed = -1.0 / nbSteps;
params = new Array ();
for (i = 0; i != nbPixels; i ++) {

  // Rajouter les paramètres des transformations des points du "pixel"

  params.push ({
    betaSpeed : (betaSpeedMin + Math.random (betaSpeedMax - betaSpeedMin))
 * Math.PI / 180.0,
    zoomSpeed : zoomSpeedMin + Math.random () * (zoomSpeedMax - zoomSpeed
Min),
    xSpeed : xSpeedMin + Math.random () * (xSpeedMax - xSpeedMin),
    ySpeed : ySpeedMin + Math.random () * (ySpeedMax - ySpeedMin)
  });

  // Rajouter les données des transformations des points du "pixel" (état du "pixel")

  for (j = 0; j != 4; j ++) {
    vertices.push (0.0); // Angle de rotation
    vertices.push (1.0); // Facteur de zoom
    vertices.push (0.0); // Translation horizontale
    vertices.push (0.0); // Translation verticale
  }
}
```

Ces formalités accomplies, les choses sérieuses commencent. Il s'agit de créer le buffer de points et le buffer d'indices.
Le buffer d'indices est le plus simple. En effet, il n'est pas nécessaire de modifier les indices durant l'animation. Le buffer peut donc être alimenté en données une fois et une seule, en spécifiant à WebGL qu'il sera statique via l'argument STATIC_DRAW de bufferData () :

```
bufferI = gl.createBuffer ();
gl.bindBuffer (gl.ELEMENT_ARRAY_BUFFER, bufferI);
gl.bufferData (gl.ELEMENT_ARRAY_BUFFER, new Uint16Array (indices), gl.STATIC
_DRAW);
```

Le buffer des points est différent. En effet, il doit être alimenté en données qu'il n'est pas nécessaire de modifier durant l'animation - les données des points X, Y, R, G et B - et en données qu'il est nécessaire de modifier à chaque étape de l'animation - l'état du « pixel » β, Z, dX et dY. La solution la plus simple serait de mettre toutes ces données dans un même tableau vertices[] et d'appeler bufferData () à chaque étape pour le copier intégralement dans le buffer associé à ARRAY_BUFFER. Toutefois, une solution plus optimisée est possible. Elle consiste à utiliser bufferData () pour copier les données des points une fois pour toutes lors de l'initialisation, puis à utiliser bufferSubData () pour copier les états des « pixels » lors de chaque étape.
Pour mettre en œuvre cette solution, il est indispensable que le buffer soit créé à sa taille définitive, c'est-à-dire incluant les données des points et les états des « pixels », car sa taille ne peut être ajustée par la suite. Cela explique pourquoi vertices[] a été alimenté avec les données et les états, et non seulement les données. Par ailleurs, il est intéressant d'indiquer à WebGL que le buffer pourra être modifié au fil des utilisations. Pour cela, l'argument DYNAMIC_DRAW est utilisé :

```
bufferV = gl.createBuffer ();
gl.bindBuffer (gl.ARRAY_BUFFER, bufferV);
gl.bufferData (gl.ARRAY_BUFFER, new Float32Array (vertices), gl.DYNAMIC_DRAW);
```

Le buffer créé et associé à ARRAY_BUFFER, il faut associer les attributs du VS à ARRAY_BUFFER. C'est l'occasion de préciser trois grandeurs à maîtriser lorsqu'il s'agit de demander à WebGL d'adresser des données. Par exemple, s'il s'agit d'utiliser vertexAttribPointer () pour que WebGL récupère dans ARRAY_BUFFER les trois nombres R, G et B correspondant à la couleur pour alimenter l'attribut A_rgb :

- offset est le nombre d'octets à parcourir depuis le début pour tomber sur le premier octet de R du premier point ;
- size est le nombre d'octets à lire pour récupérer R, G et B du premier point ;
- stride est le nombre d'octets à parcourir après B pour atteindre sur le premier octet de X du second point. [3]

In fine, cela donne :

```
gl.enableVertexAttribArray (A_xy);
gl.vertexAttribPointer (A_xy, 2, gl.FLOAT, false, 5 * Float32Array.BYTES_PER_ELEMENT, 0);
gl.enableVertexAttribArray (A_rgb);
gl.vertexAttribPointer (A_rgb, 3, gl.FLOAT, false, 5 * Float32Array.BYTES_PER_ELEMENT, 2 * Float32Array.BYTES_PER_ELEMENT);
gl.enableVertexAttribArray (A_bzdxdy);
gl.vertexAttribPointer (A_bzdxdy, 4, gl.FLOAT, false, 4 * Float32Array.BYTES_PER_ELEMENT, nbPixels * 4 * 5 * Float32Array.BYTES_PER_ELEMENT);
```

Après quelques initialisations de variables telles que le compteur des étapes, l'initialisation de l'explosion est terminée. Il est alors possible de commander le rendu de la première étape via la fonction idoine de l'objet window. Comme mentionné plus tôt, il faut écarter l'idée de recourir à setTimeout () ou setInterval () pour lui préférer requestAnimationFrame (), fonction nettement plus précise. Une astuce consiste à ne pas fournir la fonction nextStep () mais une fonction anonyme ad hoc pour optimiser le temps d'exécution, mais c'est peut-être un mythe... :

```
window.requestAnimationFrame (function () { nextStep (); });
```

Le rendu d'une étape de l'explosion

Après avoir contrôlé que l'étape courante n'est pas la dernière, la première tâche d'une étape d'une animation consiste à ... commander la suivante : requestAnimationFrame () doit être appelée immédiatement. Toutefois, ce sera ici pour plus tard afin de simplifier le code.

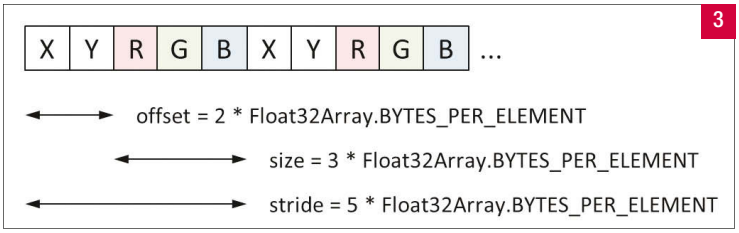
Il faut ensuite mettre à jour le buffer associé à ARRAY_BUFFER. Comme convenu, la mise à jour est limitée aux états des « pixels ». Pour cela, seule la partie de vertices[] qui les contient est utilisée lors d'un appel à bufferSubData (), cette partie étant isolée par un appel à slice (). Ce n'est certainement pas optimal, car slice () génère inutilement une copie des états dans un tableau temporaire. Toutefois, bufferSubData () ne permet pas de spécifier l'indice à partir duquel copier les données du tableau qui lui est transmis, du moins quand ce tableau est un Array. Pour optimiser, il faudrait stocker les états dans autre tableau que vertices[], sans toutefois oublier que vertices[] donne la taille du buffer (pas question donc de le faire maigrir pour autant).

```
gl.bufferSubData (gl.ARRAY_BUFFER, nbPixels * 4 * 5 * Float32Array.BYTES_PER_ELEMENT, new Float32Array (vertices.slice (nbPixels * 4 * 5)));
```

Une autre mise à jour s'impose, celle des uniformes. Il s'agit de U_mWorld, la matrice de transformation générale qui permet d'ajuster l'aspect ratio et d'appliquer une rotation aux « pixels » autour du centre de l'écran, U_a :

```
mWorld = getWorldMatrix (teta, 1.0, CANVAS_WIDTH, CANVAS_HEIGHT);
gl.uniformMatrix4fv (U_mW, false, new Float32Array (mWorld));
gl.uniform1f (U_a, alpha);
```

Les données désormais à jour, la transformation et le rendu peuvent être commandés. La surface de rendu est vidée, le blending est activé, et les



3

TRIANGLES sont transformés et rendus :

```
gl.clearColor (0.0, 0.0, 0.0, 1.0);
gl.clear (gl.COLOR_BUFFER_BIT);
gl.enable (gl.BLEND);
gl.blendEquationSeparate (gl.FUNC_ADD, gl.FUNC_ADD);
gl.blendFuncSeparate (gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA, gl.ZERO, gl.ONE);
gl.drawElements (gl.TRIANGLES, 2 * 3 * nbPixels, gl.UNSIGNED_SHORT, 0);
gl.disable (gl.BLEND);
```

Désactivé par défaut dans WebGL, le blending s'active par appel à gl.enable (gl.BLEND). L'effet consiste à mélanger la couleur d'un pixel à afficher à celle du pixel déjà affiché en pondérant l'importance des deux par son degré de transparence quantifié par la fameuse quatrième composante de la couleur, l'alpha. WebGL permet de nombreuses possibilités en la matière. Ici, on opte pour une interpolation linéaire entre la couleur de la source (le pixel du TRIANGLES à afficher) et la couleur de la destination (le pixel de la surface de rendu) :

- Rd = Rs * As + Rd * (1 - As)
- Gd = Gs * As + Gd * (1 - As)
- Bd = Bs * As + Bd * (1 - As)
- Ad = As

Ce résultat est atteint en utilisant deux fonctions : blendEquationSeparate () et blendFuncSeparate (). Sommairement :

- blendEquationSeparate () permet de préciser l'équation de combinaison entre source et destination pour les composantes R, G et B d'une part, et pour la composante A d'autre part. En l'espèce, FUNC_ADD indique que la composante X, dont la valeur est Xs dans la source et Xd dans la destination, doit prendre la valeur à Xs * Sx + Xd * Dx, où Sx est la fonction source S () appliquée à X et Dx est la fonction destination D () appliquée à X.
- blendFuncSeparate () permet de préciser les fonctions S () et D (), pour les composantes R, G et B d'une part, et pour la composante A d'autre part. En l'espèce :
  - SRC_ALPHA et ONE_MINUS_SRC_ALPHA indiquent que la composante R, G ou B doit prendre la valeur Xs * As + Xd * (1 - As) où As est la composante alpha de la source ;
  - ZERO et ONE indiquent que la composante A doit prendre la valeur As * 0 + Ad * 1 où As est la composante alpha de la source et Ad est la composante alpha de destination (autrement dit, Ad est préservée).

C'est enfin que l'animation proprement dite se déroule, pour autant que l'étape courante n'en soit pas la dernière. L'animation consiste à incrémenter l'angle de la rotation autour du centre de l'écran, décrémenter le degré de transparence (l'animation prend fin quand il atteint 0.0) et modifier l'état du « pixel » de chaque point par incrémentation de l'angle de rotation, du facteur de zoom, des composantes du vecteur de translation :

```
alpha += alphaSpeed;
teta += tetaSpeed * Math.PI / 180.0;
```

```
k = nbPixels * 4 * 5; // Sauter les données des points
for (i = 0; i != nbPixels; i ++) {
  for (j = 0; j != 4; j ++) {
    vertices[k] += params[i].betaSpeed;
    if (vertices[k] > 2.0 * Math.PI)
      vertices[k] -= 2.0 * Math.PI;
    if (vertices[k] < 0.0)
      vertices[k] += 2.0 * Math.PI;
    k ++;
    vertices[k ++] += params[i].zoomSpeed * factor;
    vertices[k ++] += params[i].xSpeed * factor;
    vertices[k ++] += params[i].ySpeed * factor;
  }
}
```

Il reste à appeler de nouveau requestAnimationFrame (), mais cet appel devrait en toute rigueur se dérouler au tout début de nextStep (), comme expliqué plus tôt :

```
window.requestAnimationFrame (function () { nextStep () });
```

La fin de l'explosion

Lorsque l'étape courante est la dernière étape, c'est le moment de faire place nette en supprimant les buffers après les avoir dissociés des attributs. En effet, l'utilisateur peut modifier le nombre de "pixels" via un champ de saisie HTML, mais les buffers ne peuvent pas être redimensionnés, si bien qu'ils doivent être recréés.

```
gl.disableVertexAttribArray (A_xy);
gl.disableVertexAttribArray (A_rgb);
gl.disableVertexAttribArray (A_bzdxdy);
gl.bindBuffer (gl.ARRAY_BUFFER, null);
gl.deleteBuffer (bufferV);
gl.bindBuffer (gl.ELEMENT_ARRAY_BUFFER, null);
gl.deleteBuffer (bufferI);
```

A la toute fin du programme, il ne reste ainsi plus qu'à éliminer les shaders et le programme :

```
function clean () {
  gl.useProgram (null);
  gl.detachShader (program, shaderV);
  gl.deleteShader (shaderV);
  gl.detachShader (program, shaderF);
  gl.deleteShader (shaderF);
  gl.deleteProgram (program);
}
```

L'exemple

Rendez-vous à l'URL suivante pour accéder à une page de test minimaliste (un simple « pixel » est affiché après l'initialisation décrite à l'instant). Vous pourrez visualiser le code et le récupérer pour travailler avec. http://www.stashofcode.fr/code/shader-explosion-de-pixels-avec-webgl-2/test.html

Pour aller plus loin...

L'explosion de "pixels" qui a été longuement décrite dans cette série d'articles n'adopte probablement pas la forme la plus optimisée qui soit, mais elle est quand même relativement optimisée. Pour plus d'informations sur l'optimisation de l'usage des buffers, il convient de se reporter aux sources suivantes :

La spécification d'OpenGL sur les buffers (https://www.opengl.org/wiki/Buffer_Object) ;

Un excellent article d'Apple (https://developer.apple.com/library/content/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/TechniquesforWorkingwithVertexData/TechniquesforWorkingwithVertexData.html) •

DevOps Rugged : intégration dans le pipeline CI

Nelly KASSEM
Consultante sécurité applicative chez **Avanade**
https://fr.linkedin.com/in/nellykassem

Pierre-Henri Gache
Manager DevOps & Agile chez **Avanade**
http://www.pierrehenrigache.com
@phgache

L'adoption des méthodes Agiles, et plus récemment de DevOps, permet de délivrer des produits de plus en plus rapidement. Si le focus est aujourd'hui donné à l'intégration continue et au déploiement automatisé, la maîtrise de la qualité et de la dette technique, ainsi que la détection et la prévention des failles de sécurité, sont encore loin d'être la priorité. Or, ces éléments sont clé dans la gestion du cycle de vie d'un produit car, ensemble, ils vont limiter au maximum la présence de nouvelles brèches de sécurité dans un SI en perpétuelle évolution.

Enjeux

Historiquement, la sécurité a toujours été une extension de l'activité de l'infrastructure. L'infrastructure intervient en fin de cycle - juste avant la mise en production - pour construire des règles de Firewall appropriées autour de l'application, les proxys protégeant généralement les clients tandis que les WAF protègent les serveurs. La sécurité des applications repose donc classiquement sur les axes suivants :

- Web application firewall (WAF) ;
- Penetration testing (Ethical Hacking) ;
- Analyse de code.

Afin d'avoir la validation de la Sécurité pour une application critique, il faut souvent ajouter à ces axes un appel à une prestation externe pour auditer le code. Vu qu'il n'existe pas de code parfait, l'audit décèlera un certain nombre de vulnérabilités de sécurité qui, à leur tour, déclencheront le pénible processus de remédiation. Celui-ci consiste en la réécriture de nombreuses lignes de code, mais aussi le plus souvent, la gestion des versions des packages open source utilisés dans les développements.

Dans un environnement DevOps qui nécessite de pouvoir livrer des versions fonctionnelles du produit en continu, il est très difficile d'intégrer ces méthodes classiques dès le début du cycle de vie du logiciel. Il est donc nécessaire d'appliquer des méthodes alternatives afin de sécuriser la livraison continue.

Pour pouvoir détecter et corriger les failles de sécurité d'un code « partiel » ou en cours de développement, il faut les traiter du point de vue de la qualité logicielle et non de celui d'y remédier par l'infrastructure. Ainsi, la faille devient un bug et l'idéal serait d'automatiser et d'industrialiser la détection de ces bugs. C'est exactement ce que l'approche DevSecOps (ou Rugged DevOps) prône.

La qualité & la dette technique

Un autre élément fondamental dans cette approche est la mesure de la qualité et donc la maîtrise de la dette technique. En effet, plus cette dette augmente et plus chaque modification de code sera périlleuse, coûteuse et impactante. Nous avons tous ou presque déjà travaillé sur une application complexe et vieillissante pour laquelle une évolution ou une correction d'anomalie relève de l'équilibrisme. Dans ce contexte, on comprend aisément que si une modification profonde doit être réalisée afin de corriger ou de prévenir des failles de sécurité, elles ne seront pas aisées voire il sera impossible d'y remédier. Il faut donc se prémunir contre ce risque et surveiller en permanence l'évolution de la qualité du code produit.

Les failles de sécurité

Les attaques d'applications Web sont maintenant fréquentes dans les infractions confirmées liées aux données (cf 2016 Verizon Data Breach Investigations Report). Pourtant, de nombreuses organisations ont du mal à mettre en œuvre un programme de sécurité applicative, car elles ne savent tout simplement pas par où commencer. Définir des politiques fondées sur l'élimination du Top 10 OWASP des vulnérabilités est un excellent point de départ - ces vulnérabilités sont largement considérées comme étant les plus susceptibles d'être exploitées, et leur correction diminuera considérablement le risque d'intrusion.

OWASP (Open Web Application Security Project) est un organisme à but non lucratif voué à fournir des informations objectives et pratiques sur la sécurité des applications. Le Top 10 OWASP représente un large consensus sur les défauts de sécurité les plus critiques pour les applications Web. Les failles sur cette liste se produisent fréquemment dans les applications Web et sont donc faciles à trouver et à exploiter. Ces erreurs sont dangereuses car elles permettent aux pirates de prendre la main sur un logiciel, voler des données ou causer l'arrêt total d'une application.

Le plus alarmant est que les attaques sur les applications Web exploitant les vulnérabilités du top 10 OWASP continuent d'émerger en production, quand bien même les scandales relatant ces incidents s'enchaînent. Cela met en évidence le gouffre existant entre tout ce qui est publié de nos jours et l'état de la sécurité applicative dans les entreprises.

La principale raison de cette totale déconnexion est l'incompréhension de ce qu'est réellement la sécurité applicative. L'analyse ponctuelle d'une poignée d'applications critiques pour les entreprises est une politique inefficace. Une sécurité applicative efficiente consiste à utiliser des outils qui évaluent en permanence les applications construites, achetées ou assemblées – depuis la phase de création jusqu'au passage en production.

Le code Open Source dans vos développements

Une version de la librairie de Google Web Toolkit contenant des failles de sécurité connues a été téléchargée plus de 18 millions de fois [1].

Pourquoi les développeurs continuent-ils à utiliser une librairie vulnérable ? Probablement parce qu'ils ne savent même pas que leur code contient cette librairie.

En 2014, l'utilisation de l'**Open Source en entreprise dépassait 80%** [2] (cette utilisation est en croissance [3]). Aujourd'hui, l'idée de développer « from scratch » des fonctionnalités standard est obsolète. Un développeur qui souhaite intégrer du SSL dans son application cherchera intuitivement une librairie de type OpenSSL à ajouter dans son code. L'équipe d'OpenSSL a aussi la même intuition. En effet, le développeur saura peut-être que son application a une dépendance (ici OpenSSL) or, il ne connaîtra probablement pas la liste des dépendances liées à la librairie en question. Il crée alors une chaîne de dépendances.

La gestion de cette chaîne – souvent inconsciente – est un vrai défi pour l'équipe de sécurité. Une liste blanche de composants conformes devient impossible à maintenir avec une utilisation d'Open Source aussi fréquente. Concrètement, la gouvernance de l'Open Source passe entre les mailles :

- **50-60%** des entreprises n'ont pas de politique officielle pour la gestion d'OSS ; [2][3]
- **33%** n'ont pas de moyen pour communiquer les problèmes de sécurité liés au OSS ;
- **47%** n'ont pas de politique officielle pour tracer l'usage d'OSS.

La gouvernance, c'est *Crawl, Walk, Run :*

- **Comprendre** : maîtriser ses librairies commence par connaître quels composants Open Source nous utilisons ;
- **Détecter** : est-ce que les composants Open Source que nous utilisons présentent un risque ?
- **Remédier** : comment peut-on réagir rapidement en cas de problème ?
- **Prévenir** : comment peut-on éviter l'utilisation des librairies vulnérables en cours de développement ou avant la mise en production ?
- **Optimiser :** améliorer l'efficacité en faisant des meilleurs choix sur les Open Source utilisés très tôt en cours de développement.

Quels outils pour quels besoins ?

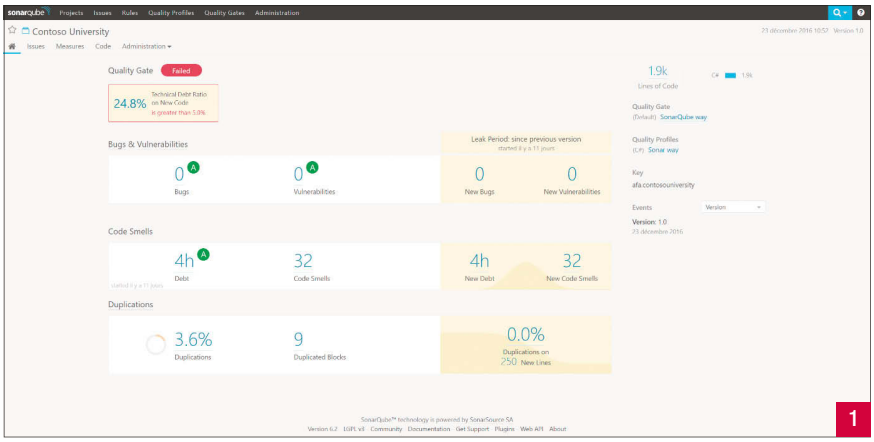
Aujourd'hui, il n'existe pas un seul outil capable d'analyser la qualité du code et de détecter les failles de sécurité. Il faut donc composer avec divers éditeurs afin d'avoir une solution capable de répondre à tous les besoins. On retrouve ainsi trois grandes catégories d'outils avec des capacités particulières :

- Qualité du code : l'objectif est de maîtriser l'évolution de la dette technique et donc d'assurer la maintenabilité du produit dans le temps ;
- Sécurité du code produit : permet d'analyser et de détecter les éventuelles failles de sécurité contenues dans le code source du produit ;
- Failles et vulnérabilités des packages : comme vu précédemment, la majorité du code est issue des packages ajoutés au projet. Il faut donc être en mesure de valider le fait que les versions utilisées ne comportent aucune faille critique.

Il est important de noter que tous ces outils ont un principe en commun : ils utilisent des règles pour calculer leurs métriques. Or, la qualité de la mesure dépendant du choix et de la pertinence de ces règles. Il y a aura donc dans le cadre d'un projet de mise en place de ces outils un travail encore plus important consistant à qualifier et créer les différentes règles utilisées par chaque outil.

Nous allons maintenant vous présenter une sélection d'outils qui répondent à un des besoins exprimés ci-dessus. Il existe d'autres alternatives mais ceux présentés ci-dessous nous semblent être les plus pertinents et les plus performants au moment où nous écrivons ces lignes.

Dernier point important : tous les outils sont multi technologies. En effet, ce genre de démarche doit s'inscrire au niveau de l'entreprise et aujourd'hui plus une seule n'utilise que du .Net ou du Java par exemple. Ce critère a donc été déterminant.



1

SonarQube

Il n'est plus nécessaire de présenter cet outil car il est devenu une référence en termes d'analyse et de mesure de la qualité de code. [1]

Les deux notions les plus importantes à retenir sont la note du projet et les « quality gates ». La première se base de la méthode SQALE pour évaluer la santé du code source. En prenant en compte divers indicateurs tels que le taux de couverture du code par les tests, le nombre de bugs détectés ou encore les duplications de code, SonarQube calcule une note valable à un instant donné. La seconde notion permet de maîtriser la dérive de cette note dans le temps. En fixant des limites sur l'évolution de différents indicateurs, les « quality gates » permettent de maîtriser la dette technique au fil des versions.

Checkmarx

Cet outil fonctionne de manière relativement similaire au précédent si ce n'est l'orientation sécurité de ses règles. En analysant le code source, Checkmarx est capable de détecter les failles de sécurité contenues dans ce dernier et ce de manière automatisée.

Il est donc ensuite possible de fixer des limites par sévérités afin de garantir qu'aucune faille critique ne sera livrée par exemple. Cet outil permet également de générer des rapports très complets et détaillés qui permettront aux équipes sécurité de valider si besoin le déploiement d'une nouvelle version.

Sonatype Lifecycle

Ce dernier outil a une approche un peu différente car il mesure la qualité des packages (NuGet, npm, bower, …) utilisés par l'application. Pour cela, il va analyser chaque package et

[1] https://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html
[2] Source: CIO.com 2014,
[3] North Bridge / Black Duck 2016 the Future of Open Source Survey of 1,300 developers, architects, and leaders,

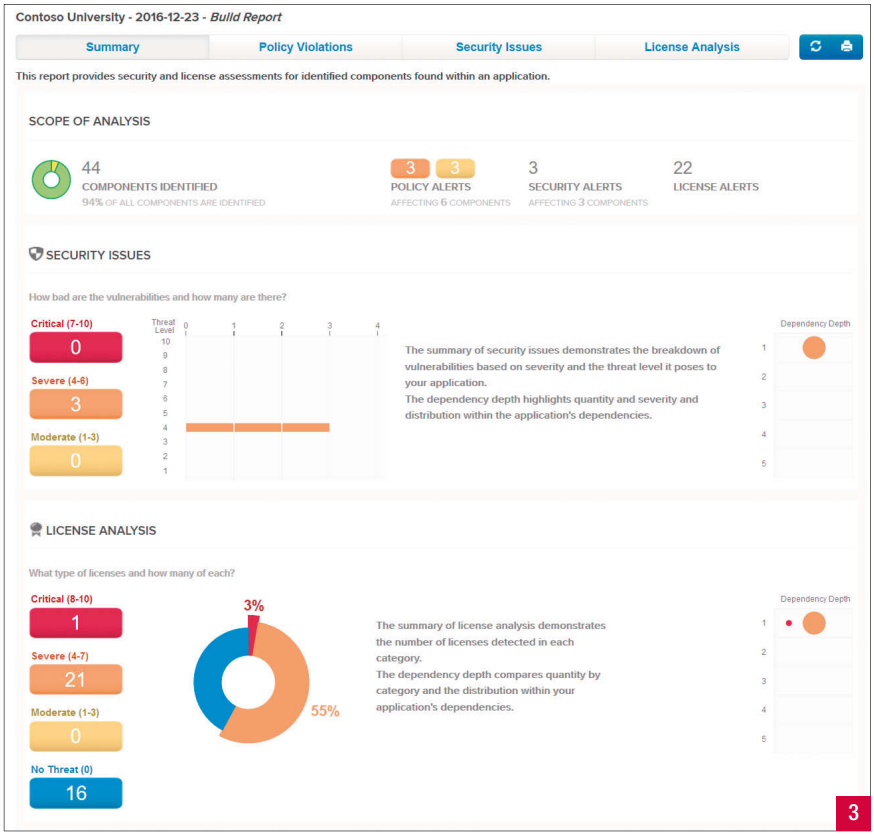
valider si la version utilisée est bien celle originale et si elle ne comporte aucune faille. [3]
Il est alors facile de détecter le code potentiellement dangereux inséré dans une librairie ou de changer de version pour une autre librairie plus sécurisé.

Au sein du processus d'intégration continue

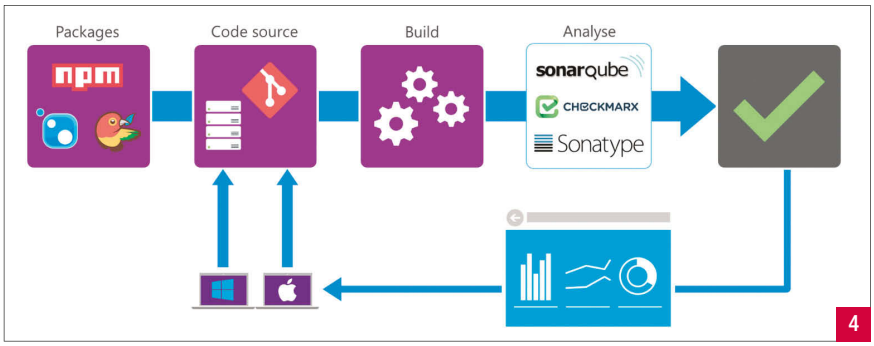
Afin d'être efficace, cette démarche doit s'inscrire dans la durée et permettre de mesurer en continu les dérives. Le pipeline de build est donc l'endroit idéal pour installer et configurer les différents analyseurs. [4]
Dans le cadre de cette démarche, nous allons utiliser le moteur de build de Visual Studio Team Services et les extensions disponibles. Au moment où j'écris ces lignes, seuls SonarQube et Checkmarx disposent d'une extension dans le marketplace. Celle pour Lifecycle est en préparation et devrait voir le jour en 2017.
La première étape consiste à lancer l'analyse Checkmarx. Etant donné que cet outil effectue une analyse du code source, il est important de le positionner en premier. Ainsi, aucune restauration de packages, notamment ceux contenant du code, ne viendra perturber l'analyse.
La seconde étape consiste à encadrer la compilation et les tests par les deux tâches SonarQube. La première a pour objectif d'initialiser l'analyseur tandis que la seconde va transmettre toute les informations collectées pour analyse.
Ces deux premières extensions sont simples à configurer et ont un fonctionnement similaire : elles ont besoin d'un endpoint contenant l'adresse du serveur, les informations d'authentification et le couple nom / identifiant du projet. Avec ces indications, le plugin peut se connecter et transmettre les informations nécessaires au serveur.
La dernière étape consiste à lancer l'analyse Lifecycle. Cette analyse vient en dernier car elle va analyser tous les packages présents dans le répertoire de build. Ainsi, même si certains packages sont issus du contrôleur de code source, ils n'échapperont pas à l'analyse. En l'absence d'extension, il faut utiliser les lignes de commandes fournies avec Lifecycle.

Analyse des résultats et corrections

A l'issue du build, il est possible de récolter tous les indicateurs provenant des différents outils. En fonction des éventuels problèmes rencontrés, il sera alors possible d'obtenir le détail de chacun. Tous les outils présentés proposent une interface Web permettant d'obtenir le détail nécessaire à la correction des anomalies.
Tous les éditeurs proposent également une extension pour votre IDE permettant d'obtenir le résultat avant le passage par l'usine d'intégration continue. SonarLint par exemple, qui est l'extension de SonarQube, permet aux développeurs de détecter et de corriger les erreurs avant le commit et donc de gagner du temps car il n'est plus nécessaire d'attendre l'analyse souvent effectuée de nuit pour découvrir le résultat.



3



4

CONCLUSION

L'automatisation et l'industrialisation de toute la chaîne de production logicielle est devenue un enjeu majeur aujourd'hui pour toutes les entreprises produisant du software. Or, si cette automatisation n'inclut pas l'ensemble des outils requis, des étapes manuelles subsisteront, ce qui aura pour effet d'augmenter le Time To Market. De plus, une approche automatisée et systématique sera toujours plus efficace et pérenne dans le temps. Avec l'approche « SecDevOps » ou « Rugged DevOps », vous avez la possibilité d'intégrer au plus tôt l'analyse et la correction des différentes failles présentes dans tout développement. Et en plus, votre production logicielle sera de qualité. Vous pourrez aussi alléger les dispositifs de contre mesure au niveau de l'infrastructure, vous permettant ainsi de réaliser des économies. •

Introduction à Rust

Partie 3

Damien Lecan,
Directeur Technique,
SQLI Nantes

Après avoir vu comment installer le compilateur Rust et Cargo, puis écrit quelques lignes de code dans la seconde partie, nous allons continuer dans cette troisième partie à faire évoluer notre programme en découvrant de nouvelles caractéristiques de Rust.

Avant de nous lancer, rappelez-vous qu'une nouvelle version de Rust est publiée toutes les six semaines ; il est donc très probable que la version que vous avez installée est déjà obsolète.
Mettez à jour Rustup lui-même, puis ensuite Rust :

```
$ rustup self update // mise à jour de Rustup
...
$ rustup update // mise à jour de tous les canaux de Rust installés
```

Pensez à effectuer régulièrement ces opérations : une version à jour de Rust vous donne accès à de nouvelles fonctionnalités et une meilleure compatibilité avec les librairies de la communauté.

Structs et traits

Notre programme des deux premières parties était structuré de manière procédurale. Nous allons lui donner une allure plus lisible et plus encapsulée, pour ne pas dire plus "objet". Rust n'a pas pour autant des "classes" comme dans d'autres langages, mais des *structs* et des *traits*.
Comme leur nom l'indique, les *structs* sont des structures qui permettent de stocker un ensemble de données complexes ; on dira qu'une *struct* définit un nouveau type. Notez qu'il n'y a pas de constructeur ou d'accesseurs à proprement parler comme dans d'autres langages, Rust reste très simple ici. Faisons évoluer notre programme en introduisant une *struct* :

```
struct Division {
    numerateur: i32,
    denominateur: i32,
}
```

La structure peut-être ensuite instanciée et ses propriétés manipulées :

```
let une_division = Division {numerateur: 10, denominateur: 12};

println!("Une division ({} - {})", une_division.numerateur, une_division.denominateur);
```

Continuons notre logique d'encapsulation en ajoutant des méthodes à notre structure. Ici, la syntaxe de Rust sort vraiment de l'ordinaire. En effet, les méthodes ne sont pas ajoutées directement dans le corps de la *struct*, mais déclarées où vous le souhaitez dans votre programme avec le mot-clé impl.

```
impl Division {
    fn calculer(&self) -> i32 {
        match self.denominateur {
            0 => panic!("Division par 0"),
            1 => self.numerateur,
            _ => self.numerateur / self.denominateur
        }
    }
}
```

Le code de la méthode calculer est repris de la méthode calculer_division vue dans la précédente partie de cet article.

Il est tentant de penser qu'on peut faire n'importe quoi, comme en JavaScript, à savoir rajouter des méthodes à n'importe quelle *struct*, écrite par vous ou fournie par le langage. Par exemple, en JavaScript, on peut rajouter des méthodes au prototype String et les utiliser comme des méthodes du langage (cette capacité est notamment utilisée par tous les *polyfills*).
En Rust, il est impossible d'ajouter des méthodes à n'importe quelle *struct* ; les impl doivent être déclarés dans le même module (crates en Rust) que la *struct*. Mettez à jour votre programme Rust :

- Déclarer le nouveau type Division à l'aide d'une *struct ;*
- Remplacer la déclaration de la méthode statique calculer_division par une méthode calculer qui implémente le type Division (attention à la signature de la méthode calculer) ;
- Remplacer l'appel à la méthode calculer_division par l'instanciation du type Division avec le paramètre qui vient de l'analyse de la ligne de commande, puis calculer le résultat et l'afficher.

A ce stade, vous devez avoir un programme qui compile et s'exécute, ce que vous pouvez vérifier à l'aide de la commande cargo run 4 (code complet https://git.io/v1XJL).
S'il n'existe pas de "vrai" constructeur en Rust, les us et coutumes du langage encouragent cependant l'encapsulation de l'opération d'instanciation d'un type au sein d'une fonction statique déclarée dans l'implémentation de la *struct* elle-même, nommée par convention new et qui retourne évidemment une instance de votre type. Ce qui donne pour Division :

```
impl Division {
    fn new(x: i32, y: i32) -> Division {
        Division {
            numerateur: x,
            denominateur: y,
        }
    }
    ...
}
```

Pensez à remplacer l'instanciation manuelle de Division dans votre programme par l'usage de la fonction new (code complet https://git.io/v1XLo).
Outre les *struct*, Rust propose un autre mécanisme qui permet d'indiquer au compilateur les fonctionnalités qu'un type doit obligatoirement fournir : les *traits*. Par exemple, si nous souhaitons contraindre la façon dont une Division doit déclarer le symbole qu'elle représente, nous pouvons déclarer le *trait* suivant :

```
trait HasSymbol {
    fn symbol(&self) -> String;
}
```

Les *traits* ressemblent aux interfaces Java ou C#, avec globalement les mêmes caractéristiques :

- Ensemble de fonctions que le type doit obligatoirement redéfinir ;

- Certaines fonctions peuvent avoir une implémentation par défaut ;
- Un *trait* peut implémenter un autre *trait*, et donc "hériter" de ses caractéristiques (concrètement, il faudra implémenter les deux *traits* sur votre type).

Implémentons notre *trait* pour la Division :

```
impl HasSymbol for Division {
  fn symbol(&self) -> String {
    "/".to_string() // "/" est du type &str, qu'il faut donc convertir
  }
}
```

Notez ce point important, impossible en Java par exemple : implémenter un trait pour un type est une section de code déclarée *a posteriori* de la déclaration de ce type, ce qui veut dire qu'un nouveau *trait* peut enrichir retrospectivement un type existant (avec quelques restrictions cependant). Désormais, il est possible de tracer le symbole de la Division, juste après sa construction, comme par exemple :

```
let division = Division::new(numerateur, 2);
println!("Symbole : {}", division.symbol());
```

Le résultat d'exécution de votre programme est désormais enrichi du symbole de la Division (code complet https://git.io/v1X4l) :

```
$ cargo run 2
   Running `target/debug/division 2`
Symbole: /
Résultat : 1
```

Encore quelques concepts à maîtriser par le Rustacéen qui débute

Vous vous en souvenez peut-être, je vous avais promis dans la première partie de cet article de la sueur, en particulier avec la gestion de la mémoire : elle n'est ni à la charge du développeur, ni à la charge d'une machine virtuelle et de son ramasse-miette. Il est temps de mettre les mains dans le cambouis et de transpirer. Accrochez-vous ! Reprenons le code décrit plus haut, et ajoutons-y une instruction qui paraît anodine :

```
let division = Division::new(numerateur, 2);
let division2 = division; // <= instruction anodine :-)
println!("Symbole : {}", division.symbol());
```

Que se passe-t-il à la compilation (cargo build) ?

```
error[E0382]: use of moved value: `division`
 --> src/main.rs:48:30
  |
47 |     let division2 = division;
  |         --------- value moved here
48 |     println!("Symbole : {}", division.symbol());
  |                              ^^^^^^^^ value used here after move
  |
  = note: move occurs because `division` has type `Division`, which
  does not implement the `Copy` trait
```

Le compilateur Rust vous indique que la valeur pointée par la variable division est **déplacée** (*moved*). En effet, une valeur ne peut être référencée que par une seule variable dans tout notre code. C'est le concept de possession (*ownership*), contrôlée à la lettre par le compilateur, qui va garantir la sécurité et la robustesse d'exécution d'un programme Rust. Nous avons déplacé la possession de la valeur de division vers division2, par conséquent, division ne peut plus être utilisée.

Qui dit "possession", dit "emprunt" (*borrowing*) ou "copie" et pour illustrer l'une ou l'autre des solutions, nous allons introduire une nouvelle fonction statique et l'utiliser (code complet https://git.io/v1X2m):

```
fn display_symbol(division: Division) {
  println!("Symbole: {}", division.symbol());
}
...
let division = Division::new(numerateur, 2);
display_symbol(division); // <= à la place du 'let division2 = ...'
```

Ce code ne compile pas et vous obtenez un message d'erreur très similaire au précédent. Trois solutions pour résoudre cette erreur de compilation s'offrent à nous :

1/ Ne plus utiliser la variable division après l'appel à la méthode display_symbol ;

2/ Copier toute la valeur de la variable ;

3/ Permettre l'emprunt de la variable et utiliser sa référence.

La solution 1/ est une façon de mettre le problème sous le tapis ou de le repousser, ce n'est pas vraiment une solution (même si le programme compilera effectivement ;-)

Copier toute la valeur de la variable (solution 2) est une façon intéressante de résoudre le problème. Dans cette solution, la valeur de la variable est entièrement copiée lors de l'appel à la fonction. Notez bien qu'il n'y a donc plus de lien entre la variable déclarée avant l'appel de la fonction et la variable utilisée dans la fonction.

Pour déclarer un type "copiable", il suffit d'implémenter le *trait* Copy, ainsi que le *trait* Clone dont hérite Copy. Le plus simple, est d'utiliser l'implémentation automatique de certains *traits,* proposée par l'attribut #[derive(...)] (documentation : bit.ly/2hASRHt), à appliquer directement sur un type :

```
#[derive(Copy, Clone)]
Division {
  ...
```

Avec cet ajout, notre programme compile de nouveau (code complet https://git.io/v11tL). Cependant, si cette solution peut sembler magique et simple (et de plus proposée par le compilateur), elle a aussi ses limites :

Un type peut implémenter Copy uniquement si l'ensemble de ses composants implémente aussi Copy (dans notre exemple, i32 implémente Copy, donc Division peut implémenter Copy). Pour les Javaistes, on retrouve ici une propriété équivalente dans l'esprit aux objets sérialisables …

Un certain nombre de types n'implémentent pas Copy ; par oubli du développeur ou par incompatibilité technique. Par exemple, les Strings ne sont pas compatibles avec Copy. La troisième et dernière solution s'applique donc à tous les autres cas. Elle s'appuie sur le principe de l'emprunt, à savoir qu'on va passer à notre fonction non pas la valeur d'une variable, mais une référence vers la valeur de cette variable.

Concrètement, il faut faire évoluer la signature de notre fonction d'affichage pour permettre l'emprunt et l'usage de référence :

```
fn display_symbol(division: &Division) { // <= Notez le & ici
  ...
}
...
let division = Division::new(numerateur, 2);
display_symbol(&division); // <= Notez le & ici
```

La fonction display_symbol attend une référence vers un type Division, identifiée par le symbole & et nous pouvons passer la référence vers une valeur en ajoutant le même symbole devant le nom d'une variable (code com-

plet https://git.io/v11ml). Cette solution est la plus universelle, au prix d'une complexité intellectuelle plus élevée.

Finalement, grâce à ces différents mécanismes, le cycle de vie de toutes les données référencées par des variables, est connu dès la compilation de votre programme, ce qui garantit une bonne gestion de la mémoire et donc une certaine robustesse. En revanche, elle vous garantit aussi quelques "combats" contre le *borrow checker* du compilateur Rust.

CONCLUSION

Au cours de ces trois articles d'introduction à Rust, nous avons abordé les outils et les concepts les plus importants du langage : *struct*, *trait*, *pattern matching*, *enum*, possession (*ownership*), emprunt (*borrowing*), notions de programmation fonctionnelle … Ces connaissances devraient être suffisantes pour écrire vos premiers programmes avec Rust.

Vous découvrirez par vous-même des concepts plus avancés du langage comme les *lifetimes*, les types génériques, les modules et les *crates*, les macros, les tests, les itérateurs, la concurrence, la gestion des erreurs … La lecture du livre "The Rust programming language" sera donc à ce propos un bon point de départ pour vous : https://doc.rust-lang.org/book/. Enfin, tous ces concepts pourraient faire l'objet d'articles dédiés dans Programmez!, avis donc aux amateurs !

•

EmbedXcode : comment programmer un Arduino à l'aide d'Xcode sous Mac Partie 1

• Renaud Pinon

La fondation Arduino ne se contente pas de créer le design des cartes homonymes : elle fournit aussi un éditeur de code relativement basique, Arduino IDE, et ce, sur les trois plateformes principales que sont Windows, macOS et Linux. Si l'on peut y voir un bel effort pour rendre la programmation de ces cartes accessible au plus grand nombre, le développeur habitué aux éditeurs modernes ressentira un grand désarroi face à cet outil : pas de véritable coloration syntaxique (si ce ne sont quelques mots prédéfinis), pas d'auto-complétion, une gestion des fichiers archaïque, … Bref, un vrai voyage temporel dans les années 90 ! Une question se pose alors : mais pourquoi le Mac User ne pourrait-il pas simplement programmer son Arduino à partir d'un éditeur moderne comme Xcode, que tout développeur d'applications macOS ou iOS maîtrise déjà ?

Présentation d'*EmbedXcode*

EmbedXcode est justement un modèle de projet pour Xcode permettant le développement sur les cartes Arduino. Il est développé par Rei VILO et disponible en téléchargement à l'adresse http://embedxcode.weebly.com/.

Le téléchargement est basé sur le principe de donation, avec une version gratuite, mais aussi deux versions « + », *Personal* et *Commercial*, pour des sommes conseillées à respectivement 29 et 99 .

N'ayons pas peur des mots : utiliser *Xcode* pour programmer un *Arduino* est une véritable libération. Les avantages sont nombreux puisqu'enfin toutes les fonctionnalités de développement modernes sont disponibles : une véritable coloration syntaxique, une gestion des fichiers efficace grâce à l'arborescence de gauche, la proposition de code, l'auto-complétion, la possibilité d'aller directement à la définition d'une fonction ou d'une variable grâce au raccourci clavier *CMD* + clic, la gestion du *versioning* avec *GIT*, etc.

Pré-requis

Avant d'installer le modèle de projet, vous devrez bien évidemment avoir installé *Xcode* sur votre machine. Je ne rentrerai pas dans les détails de l'installation : il suffit de le télécharger gratuitement sur le *Mac App Store*. Attention : le téléchargement peut être long, plusieurs giga octets vous attendent…

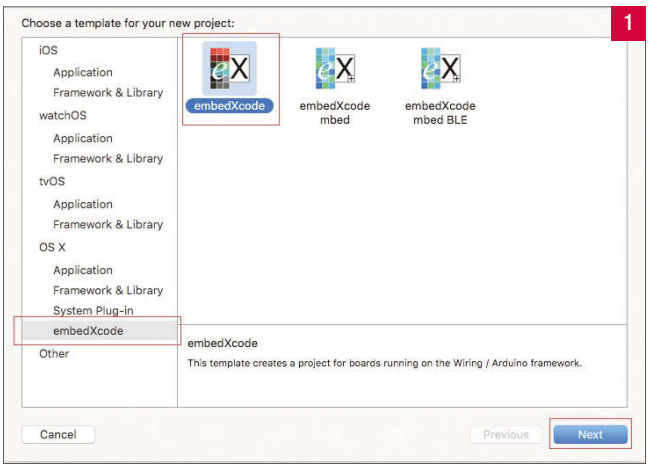
Une fois *Xcode* présent sur votre machine, vous serez tout de même dans l'obligation d'installer *Arduino IDE*. Oui je sais, c'est un peu idiot alors qu'on voulait s'en passer, mais c'est tout simplement car cet éditeur contient les fichiers standard qu'*EmbedXcode* ira chercher au moment de la compilation. Si *Arduino IDE* n'est pas déjà présent sur votre ordinateur, vous pouvez télécharger la version *macOS* sur la page : https://www.arduino.cc/en/Main/Software. Reportez-vous au site pour plus de détails sur l'installation. Une fois installé, lancez *Arduino IDE* au moins une fois.

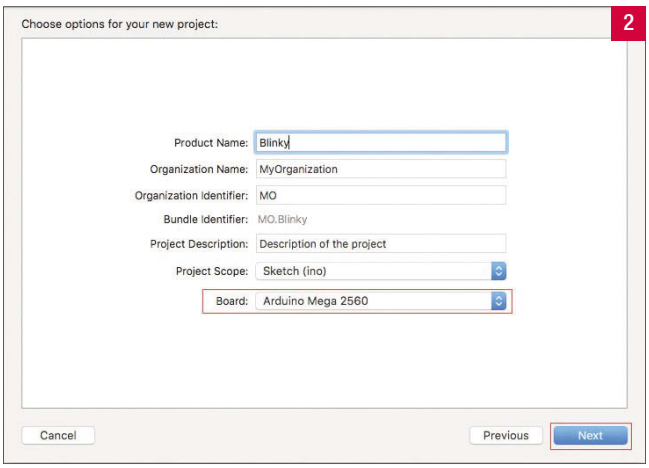
Enfin, pour télécharger *EmbedXcode*, allez sur la page http://embedxcode.weebly.com/ puis cliquez sur le menu *Download* en haut à droite. Une page vous permettra de choisir le type de version (gratuite ou avec donation). Je vous conseille la version gratuite pour le moment. Toutefois, si à l'avenir vous l'utilisez régulièrement, je vous invite fortement à considérer l'achat de la version personnelle ou commerciale : n'oubliez pas qu'un développeur ne programme pas toujours simplement pour la gloire et la postérité, mais aussi parce qu'il a besoin de manger ;)

En ce qui concerne l'installation d'*EmbedXcode*, je vous renvoie à la documentation présente sur le site en tant que fichier *.pdf*, ou disponible gratuitement sur le *Book Store* d'*Apple*.

Création du premier projet

Si le téléchargement et l'installation du fichier *.pkg* d'*embedXcode* se sont déroulés correctement, un nouveau type de projet est désormais disponible dans *Xcode*. Vérifions cela de suite : lancez *Xcode* et créez un nou-





veau projet (soit à partir du lien de l'écran d'accueil, soit en faisant le menu *File / New / Project…*). [1]

Dans la section *OS X* à gauche, une nouvelle catégorie intitulée *embedXcode* est présente : sélectionnez-la, puis dans la partie droite choisissez la première icône : « *embedXcode* ». Enfin, cliquez sur le bouton *Next*. [2]

Définissez ensuite le nom du projet (ici : *Blinky*), le nom de votre organisation et le type d'*Arduino* vers lequel vous enverrez le code (un *Arduino MEGA* pour ma part). Enfin, cliquez sur le bouton *Next* pour choisir le dossier où seront enregistrés les fichiers.

Maintenant que notre projet est créé, il faut définir le port *USB* correspondant à notre *Arduino*. Pour connaître le nom du port dans le système, une solution très simple existe : lancez *Arduino IDE* (oui, le logiciel dont on voulait se passer), branchez votre *Arduino* en *USB* à votre *Mac* et allez dans le menu *Outils / Port*.

Repérez le port correspondant à votre Arduino et notez très précisément la ligne */dev/XXX.usbmodemXXXX* (excepté les termes entre parenthèses). [3]

Retournez ensuite dans *Xcode*, cliquez sur le nom du projet tout en haut à gauche dans l'arborescence, sélectionnez la cible « *All* », puis cliquez sur « *Build settings* ».

Dans le paramètre *BOARD_PORT / Debug*, vous verrez la valeur */dev/tty.usbmodem1**. Remplacez cette valeur par celle trouvée dans *Arduino IDE* (dans mon cas : */dev/cu.usbmodem1421*). [4]

Ensuite, afin d'y voir plus clair dans la liste hiérarchique de gauche, je vous conseille de replier les dossiers dont on ne se servira pas. Il s'agit de *Sketchbook*, *Resources*, *Configurations*, *Makefiles* et *Utilities*. Vous verrez tout en haut de la hiérarchie le fichier principal intitulé « *Blinky.ino* » : c'est lui qui contient notre programme et comporte pour le moment, en plus de diverses instructions pour le pré-processeur, les fonctions *setup()* et *loop()* totalement vides. Si vous affichez le fichier, vous constaterez qu'aucune coloration syntaxique n'est faite par *Xcode* ! C'est parce qu'il faut compiler le programme une première fois avant que la coloration syntaxique et l'auto-complétion ne soient disponibles. Cliquez donc sur le bouton *Play* en haut à gauche ou faites le menu *Product / Build* pour lancer la compilation et l'envoi vers la carte Arduino. Si tout va bien, vous verrez le clignotement familier des LEDs *RX* et *TX* de votre *Arduino* correspondant à l'envoi du programme.

A ce stade, une fenêtre *Terminal* se lance automatiquement : il s'agit de la communication série avec l'*Arduino*. Pour le moment rien ne s'y affiche, mais plus tard nous écrirons des choses grâce à l'instruction *Serial.println()* que vous connaissez tous.

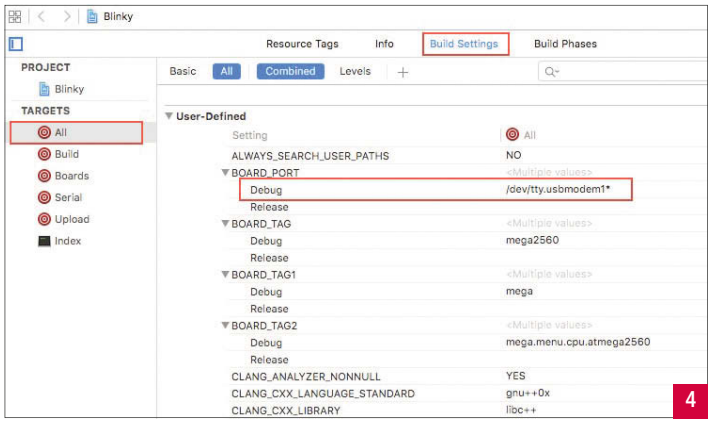
Encore une fois, si tout va bien, le contenu du fichier *Blinky.ino* doit présenter la coloration syntaxique du code. Si ce n'est pas le cas je vous conseille simplement de fermer *Xcode* et de le relancer, tout devrait rentrer dans l'ordre. A noter que c'est le seul moment où vous aurez ce genre de problème : vous constaterez par la suite que lorsque l'on crée des nouveaux fichiers la coloration et l'auto-complétion fonctionnent immédiatement. [5]

Modifions maintenant le fichier *Blinky.ino* pour enfin faire clignoter notre LED. Copiez les lignes suivantes à la place des fonctions *setup()* et *loop()* déjà existantes :

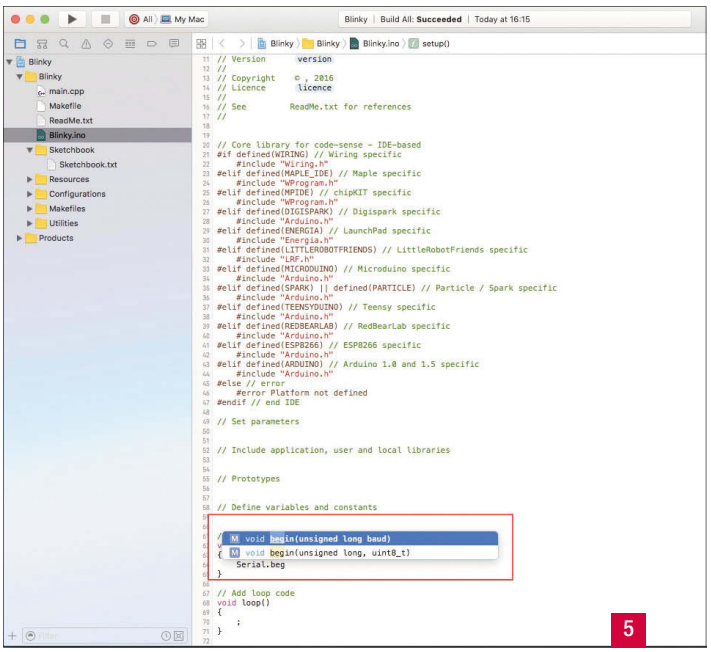
```
// Add setup code
void setup()
{
  Serial.begin(9600);      // init connexion série.

  pinMode(13, OUTPUT); // Définit la broche 13 (= LED intégrée) en sortie.
}

// Add loop code
void loop()
{
  digitalWrite(13, HIGH);  // Eteint la LED
  delay(1000);             // Attend une seconde
  digitalWrite(13, LOW);       // Allume la LED
  delay(1000);             // Attend une seconde
  Serial.println("Blink ;)"); // Ecrit "Blink ;)" dans le moniteur série.
}
```

Envoyez le programme à l'*Arduino* grâce au bouton *Play* en haut à gauche (ou, encore une fois, le menu *Product / Build*) : une fois le programme téléversé, la *LED* de votre *Arduino* doit alterner une seconde

3



4



5

allumée et une seconde éteinte. De plus, la fenêtre du terminal imprime désormais toutes les deux secondes la ligne « *Blink ;)* ». [6]

Structure du projet

Vous aurez constaté que de nombreux fichiers et dossiers se trouvent dans l'arborescence de gauche, attardons-nous donc un peu sur leurs rôles. Tout d'abord le fichier *main.cpp* est bien évidement le point d'entrée du programme. Toutefois, je vous déconseille de le modifier puisque tout y est déjà calibré au iota près afin d'inclure notre fichier principal *Blinky.ino*. Comme évoqué précédemment, ce fichier *.ino* contient une ribambelle d'instructions destinées au pré-processeur :

```
// Core library for code-sense - IDE-based
#if defined(WIRING) // Wiring specific
  #include "Wiring.h"
//[...lignes omises par souci de clarté...]
#elif defined(ARDUINO) // Arduino 1.0 and 1.5 specific
  #include "Arduino.h"
#else // error
  #error Platform not defined
#endif // end IDE
```

Encore une fois, je vous conseille de ne pas les toucher, ni même de les déplacer dans un fichier .h, car sans cela, la complétion du code ne fonctionnera plus. Pour référence ultérieure dans cet article, nous nommerons ces lignes les « *#define magiques* » ;)

6

Dernier fichier à la racine, le fichier *Makefile* (sans extension) contient divers paramètres relatifs aux chemins de compilation, mais aussi aux chemins des bibliothèques que le compilateur doit considérer au moment de la création du programme. Nous y reviendrons plus en détail lorsque nous parlerons de l'utilisation des bibliothèques, de même que nous parlerons du dossier *Sketchbook* permettant d'ajouter les bibliothèques utilisateur au projet.

Enfin, vous n'aurez sans doute aucune interaction avec les autres dossiers de l'arborescence : *Resources* pointe vers les fichier en-tête de base d'*Arduino IDE* (d'où l'importance de l'installation de ce dernier), *Configurations* contient des fichiers *.xconfig* énumérant les paramètres de configuration pour chaque type d'*Arduino*, *Makefiles* contient les arguments de compilation pour chaque type de microcontrôleur, tandis qu'*Utilities* contient des ressources graphiques et des programmes utilisés en interne par *embedXcode*.

Et pour en finir avec la structure des projets, parlons des cibles (*Target*) de compilation. Vous constaterez que plusieurs destinations existent. Voici leurs rôles :

- All (choix par défaut) : efface les fichiers déjà compilés, compile, envoie le programme à l'*Arduino* et ouvre le terminal pour la communication série ;
- Build : efface les fichiers déjà compilés, compile mais n'envoie pas le programme à l'*Arduino* ;
- Boards : liste les modèles de cartes supportées dans la console des erreurs et avertissements ;
- Serial : ouvre une fenêtre *Terminal* avec une connexion série vers l'*Arduino*. Ne compile rien et n'envoie pas de nouveau programme à l'*Arduino* ;
- Upload : ne re-compile pas le programme mais l'envoie à l'*Arduino* tel qu'il a été précédemment compilé. N'ouvre pas de fenêtre *Terminal*.

Ces options ne sont intéressantes que si votre projet prend de l'ampleur et que vous avez besoin de gagner du temps à la compilation, ou par exemple si vous souhaitez envoyer votre programme à toute une batterie d'Arduino sans avoir à recompiler le programme à chaque fois. A noter que la version + d'*embedXcode* contient des cibles supplémentaires censées vous faire gagner encore plus de temps. •

La suite dans Programmez 205

Découvrez tangente le magazine des mathématiques

Pour mieux comprendre le monde

tangente
l'aventure mathématique

Aussi en version numérique !

Tous les deux mois chez votre marchand de journaux

tangente
l'aventure mathématique

Politique, économie, finance, jeux, musique, littérature, arts plastiques, architecture, **informatique**, physique, biologie, géographie : **les mathématiques sont partout !!!** Le magazine *Tangente* et ses hors séries vous aident à redécouvrir notre quotidien.

Les hors séries « kiosque »

4 fois par an, un hors série d'au moins 56 pages, explore un grand dossier de savoir ou de culture. **Derniers parus :** Les angles - Les graphes - Les démonstrations - Les fonctions - Maths des assurances - Maths et médecine - La Droite - Maths et Architecture - Les ensembles

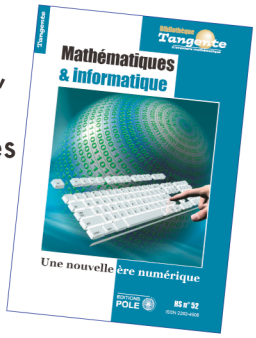
Disponibles chez votre marchand de journaux ou avec **l'abonnement PLUS.**

Vous voulez vous rendre compte de ce qu'est la consultation numérique d'un numéro de *Tangente* ?

L'accès au numéro 167 vous est offert !

Rendez-vous sur http://tangente-mag.com (identifiez-vous)

La « Bibliothèque Tangente »



Pour les lecteurs les plus curieux, les articles des hors séries de *Tangente* sont repris et complétés dans la Bibliothèque Tangente, avec de magnifiques ouvrages d'environ 160 pages (prix unitaire 20 à 22 €), richement illustrés, disponibles

- sur la boutique du site **www.infinimath.com**
- chez votre libraire
- avec **l'abonnement SUPERPLUS.**

http://tangente-mag.com

DEMANDEZ UN ANCIEN NUMÉRO de *Tangente* - Joignez juste 3 € de timbres

À adresser à *TANGENTE* - 80 BD SAINT-MICHEL - 75006 PARIS avant le 31/03/17

NOM* . . . . . PRÉNOM* . . . . .

ADRESSE* . . . . .

CODE POSTAL* . . . . . VILLE* . . . . . PAYS . . . . .

MAIL* . . . . . TÉLÉPHONE* . . . . .

ABONNÉ À PROGRAMMEZ ❒ OUI ❒ NON PROFESSION . . . . .

Atari ST : Overscan

• Sengan Baring-Gould
Docteur en Intelligence Artificielle et CEO d'Ansemond LLC. Auteur du livre Sams Teach Yourself Cocoa Touch Programming in 24 Hours (Développement d'applications sur iPhone en 24h).

Overscan ? Fullscreen ? Hardscroll ? Ces mots ont été depuis longtemps réservés aux passionnés et codeurs sur Atari ST. Il est temps aujourd'hui de vous révéler comment les développeurs de l'époque dépassaient les capacités techniques et hardware de la machine.

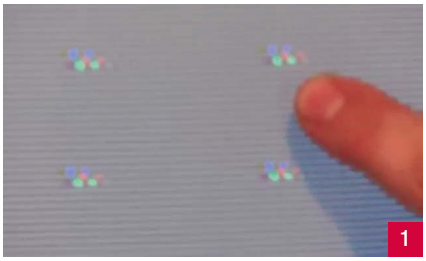
Petit retour en arrière

Il faut se souvenir qu'à l'époque - il y a donc 30 ans - l'Atari ST pouvait être branché soit à un téléviseur soit à un moniteur, tous deux utilisant un tube cathodique. Ces écrans pouvaient afficher 625, 525 ou 500 lignes au total, qui étaient balayées par un ou trois faisceaux d'électrons 50, 60 ou 71 fois par seconde respectivement. Afficher un pixel sur l'écran consiste à régler le voltage des faisceaux d'électrons quand ils ciblent ce pixel. Le système vidéo de l'Atari ST doit lire la mémoire dédiée aux graphismes à montrer sur le moniteur et régler le voltage qu'il envoie au moniteur au même moment que les faisceaux d'électrons ciblent chaque pixel de l'écran pour que l'image voulue apparaisse sur l'écran. [1]

Le système vidéo de l'Atari ST est composé de différents composants dont la principale puce dédiée est appelée le **SHIFTER**, qui est aidée par les composants dénommés la **GLUE** et la **MMU**. C'est la **MMU** qui accède à la mémoire à accès aléatoire (RAM) via le bus 16-32 bits du ST et envoie ce qu'elle lit au **SHIFTER**. Le **SHIFTER** utilise ces informations pour régler le voltage qu'il envoie au moniteur.

Parce que les téléviseurs et les moniteurs avaient des réglages différents, les côtés des images n'étaient souvent pas visibles. Pour éviter que les utilisateurs rouspètent, la plupart des ordinateurs de l'époque affichaient des bordures autour de l'image utile, c'est à dire la partie de l'écran sur laquelle on peut afficher des graphiques. Par exemple, un téléviseur Français aux normes SECAM avait 625 lignes entrelacées. C'est à dire que les faisceaux d'électrons balayaient alternativement 313 et 312 lignes de l'écran 50 fois par seconde.

L'Atari ST trichait et envoyait 313 lignes 50 fois par seconde. Un téléviseur Américain aux normes NTSC avait par contre 525 lignes entrelacées, et l'Atari ST lui envoyait 263 lignes 60 fois par seconde. Le moniteur noir et blanc d'Atari balayait environ 500 lignes environ 71 fois par seconde en mode MONOCHROME. En particulier, l'image utile apparaît dans 200 de ces lignes (basse résolution de 320x200 et moyenne résolution de 640x200). Les lignes restantes consistent en des signaux de synchronisation et surtout depuis la bordure du haut et celle du bas. Puisqu'il reste 63 lignes pour un téléviseur Américain, l'image utile commence à la 34e ligne affichée, et s'arrête à la 234e ligne. La bordure continue alors jusqu'à la 263e ligne. Par contre, pour un téléviseur Français, il reste 113 lignes, et l'image utile commence selon le modèle d'Atari ST, soit à la ligne 47, soit à la ligne 63. C'est la **GLUE** qui s'occupe de produire ces signaux de synchronisation, et qui décide quand le **SHIFTER** doit afficher une bordure et quand il doit afficher l'image utile.


1

2

3

Hypothèses

Puisque la fréquence du moniteur est programmable, on peut se demander si on peut « embrouiller » le système vidéo pour qu'il supprime les bordures, sans embrouiller le moniteur qui lui attend que les signaux venant du système vidéo soient cadencés de façon très précise pour que les pixels soient placés correctement sur l'écran. La réponse est affirmative. On peut, en passant à 60Hz (mode NTSC) à la fin de la ligne 33 et en revenant å 50Hz (mode SECAM) juste après le début de la ligne 34. Pour ce faire il suffit(!) de synchroniser son programme avec le faisceau d'électrons du moniteur. Heureusement, le compteur vidéo du système vidéo révèle l'adresse mémoire que le système vidéo lit pour envoyer les pixels de l'image utile au moniteur. Le compteur est donc synchronisé avec le faisceau d'électrons du moniteur, et peut servir pour synchroniser un programme tournant sur le 68000 avec ce faisceau d'électrons. Cela permet de changer la palette plusieurs fois par ligne pendant l'affichage de l'image (images Spectrum 512), et d'éliminer les bordures (mode Overscan). En exploitant la différence entre les différents modes vidéo, on parvient à éliminer toutes les bordures (haut, bas, droite et gauche) en changeant entre les trois modes vidéo (SECAM, NTSC, MONOCHROME) à des moments bien précis, et pour des durées bien déterminées, à chacune des 274 lignes de l'image utile sans bordures.

Voici les différents types de bordures entre un écran couleur en BASSE résolution à 60Hz [2] et un écran monochrome en HAUTE résolution à 50Hz [3].

Puisque les changements de mode vidéo requis pour générer un Overscan complet ont lieu à divers moments pendant 90% du temps machine disponible, tout programme qui fait quelque chose d'utile doit être parsemé de routines qui changent le mode vidéo à des moments bien précis. Pour créer ces programmes, il fallait coder en assembleur et prêter attention à la durée de chaque instruction. On pouvait alors diviser son programme en séquences d'instruc-

tions de la durée requise qu'on pouvait placer entre les routines changeant le mode vidéo. Un travail d'optimisation acharné donc..

Synchronisation et optimisation

Comme le système vidéo et le programme doivent être synchronisés, en général le code exécuté pendant l'Overscan n'a aucun branchement conditionnel: on veut éviter de calculer plusieurs durées par séquence d'instruction (à cause des chemins multiples à travers le code). Le code qui tourne pendant un Overscan sert donc surtout à faire des tâches répétitives, d'une façon assez similaire au code d'aujourd'hui qui est destiné à tourner sur les cartes graphiques. Il faut se rappeler qu'à l'époque, seulement les registres du système vidéo étaient documentés, mais pas leur fonctionnement interne. Tous les changements de mode vidéo ont été découverts en expérimentant avec des programmes tests. D'ailleurs, bien qu'on ait découvert les changements de mode vidéo nécessaires pour éliminer toutes les bordures, l'image obtenue n'était pas stable. Plus d'expérimentation révéla qu'un changement de résolution pour une durée courte à la fin de chaque ligne stabilisait l'image, mais on ne savait pas pourquoi ?!

La découverte de l'Overscan a permis des innovations graphiques techniques telles que le Hardscroll. Sur l'Atari STE, il suffit de changer l'adresse du début de la mémoire vidéo pour déplacer l'écran visible (multiples de 16 pixels) vers la gauche ou vers la droite, et de lignes entières vers le haut ou le bas. Mais ce n'était pas possible sur l'Atari ST. Du bon code assembleur prend environ 140000 cycles pour déplacer les graphiques d'un écran sans Overscan, et presque le double pour un écran en Overscan. 140000 cycles correspondent à 85% du temps disponible, puisqu'on ne dispose que de 160256 cycles si on veut faire des animations fluides qui tournent 50 fois par seconde.

Le Hardscroll a pour origine l'observation suivante: une ligne sans bordure consomme plus de mémoire vidéo qu'une ligne avec des bordures. Le compteur vidéo aura donc plus avancé à la fin d'une ligne sans bordure qu'une ligne avec. Il est donc possible de trouver des combinaisons de lignes avec ou sans bordures à droite ou à gauche qui permettent de spécifier l'adresse du début de la mémoire vidéo qui suit au mot près. Cela ne prend pas plus que 3000 cycles, soit presque 50 fois moins de temps : un gain immense permettant la réalisation de Scrollers en Overscan totalement fluides utilisés dans les jeux tel que **Lethal Xcess** d'*Eclipse Software Design* ou **Enchanted Lands** de *Thalion Software*

Pour la petite histoire

En 1989 je désirais passionnément faire mon propre Overscan. Après beaucoup d'expérimentations, j'ai réussi à faire un Overscan stable et un Hardscroll fluide sur Atari STF. Mais je voulais aussi comprendre comment fonctionnait le système vidéo de l'Atari, pourquoi l'Overscan était possible et pourquoi il fallait absolument le stabiliser. J'ai donc tenté de recréer ce système vidéo sur papier en utilisant tout ce que je savais du hardware, et en écrivant de nombreux programmes de test qui permettaient de décider entre toutes les hypothèses possibles. J'ai documenté les résultats de ces recherches dans mes articles publiés dans *ST Magazine* de l'époque. [4]

Un de ces résultats a été la découverte du Hardscroll 4-bits, une méthode pour décaler l'image utile de la droite vers la gauche sur un simple STF, ce qui était bien utile pour les Scrolling Horizontaux. Codeur du groupe ST CONNEXION, j'ai codé en 1991 un écran démontrant les nouvelles capacités de l'Atari ST avec une musique Soundtrack pour la méga démo **Punish Your Machine** du groupe Delta Force intitulé *Let's Do The Twist Again*.

Quelques articles ont ensuite servi à créer des ré-implémentations de l'Atari ST sur des circuits intégrés reprogrammables FPGA et pour les émulateurs Atari ST. Et cette année, un hacker plus assidu que moi, *Jorge Cwik* a démonté complètement la puce du **SHIFTER** et il a trouvé les registres internes dont j'avais démontré l'existence pour obtenir les effets que j'observais dans mes programmes de test à l'époque. Pour les détails précis des valeurs nécessaires pour obtenir un Overscan je vous renvoie donc à ces articles que vous trouverez dans les ST Magazines numéros 51, 52, 55 et 70. Le site Internet http://abandonware-magazines.org/ de *Fréderic Letellier-Cohen* en a préservé des copies que vous pouvez consulter en ligne ou en téléchargement. Vous y trouverez aussi un historique des programmeurs qui ont découvert chaque aspect de l'Overscan et du Hardscroll. Finalement, vous y trouverez aussi un outil, l'intégrateur, qui permet de calculer comment décomposer votre programme pour y insérer les changements de mode vidéo nécessaires pour obtenir l'Overscan.

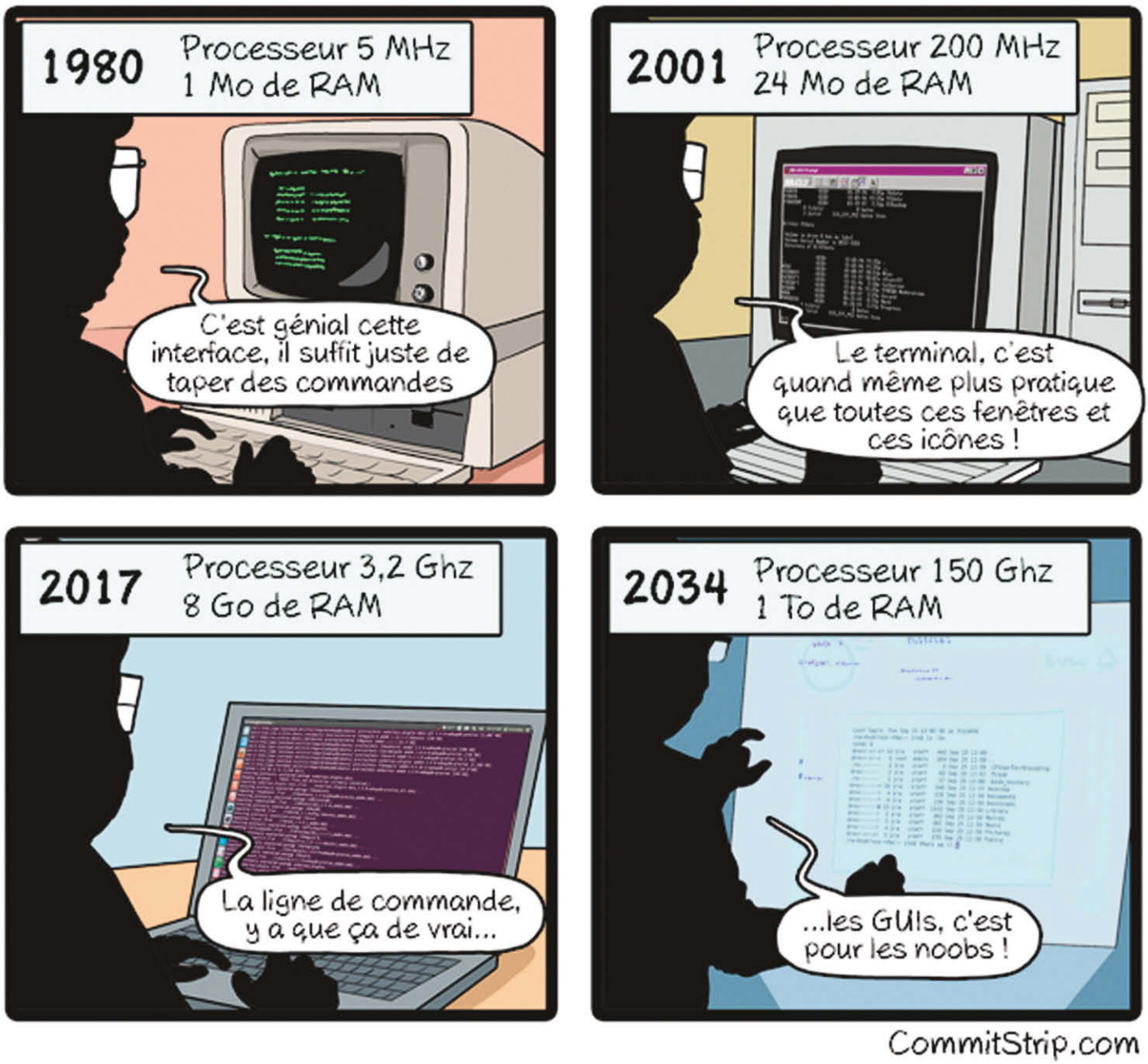
Depuis 1989, il y a eu quelques découvertes de plus sur le mode Overscan sur l'Atari ST via le monde de la Démoscene. *Troed Sångberg* (codeur du groupe SYNC) a résumé mon travail et ce qui a été découvert depuis sur son blog https://blog.troed.se/2015/12/23/overscan-and-sync-scrolling/.

•

```
*****************************************************************
* DEMONSTRATION OVERSCAN
*****************************************************************
mfp equ 40                      ; Délai mpf à partir duquel on teste.

Démonstration
    bsr CLS                     ; Effacement écran
    move.b  #mfp,mfp_Ada+3      ; Lancement du test adaptatif
    bsr Adapt
    bsr Set_Up_Hardware
    stop    #$2300
    stop    #$2300
    move.l  #VBL,$70.w
.loop addq.w    #1,d6
    cmp.b   #$B9,$fffffc02.w
    bne.s   .loop
    move.w  #$2700,sr
    rts

Set_Up_Hardware                 ; Initialisation du hardware.
    clr.l   $fffffa06.w         ; On interdit les intérruptions
    clr.l   $fffffa12.w         ; mfp.
    move.l  #RTE,$70.w          ; Reset Shifter
    stop    #$2300
    stop    #$2300
    clr.b   $ffff820a.w
    stop    #$2300
    move.b  #2,$ffff820a.w
    move.w  #$2700,sr           ; Mise en Basse résolution et en
    clr.b   $ffff8260.w         ; 50 Hz
    move.b  #$2,$ffff820a.w
    lea $fffffa00.w,a0          ; Initialisation mpf
    clr.l   $0A(a0)
    clr.b   $19(a0)
    clr.b   $1B(a0)
    clr.b   $1D(a0)
    move.b  mfp_Ada+3(pc),$1f(a0) ; délai mfp avant sa prochaine
    bclr    #3,$17(a0)          ; intérruption
    move.w  #%0010000100000000,d0
    movep.w d0,$7(a0)
    movep.w d0,$13(a0)
    move.l  #TIMERB,$120.w
    move.l  #MFP,$134.w
    rts

Adapt:  bsr Set_Up_Hardware     ; Code adaptatif qui découvre
    move.l  #.TIMERB,$120.w     ; le délai correct du timer A
    stop    #$2300              ; pour l'obtention d'un over-
    stop    #$2300              ; scan
    move.l  #VBL,$70.w
.again  moveq   #1,d2
.again2 moveq   #0,d1
.attente_fin_écran              ; Attente que les 199 lignes
    tst.b   d1                  ; d'overscan aient été montrés
    beq.s   .attente_fin_.cran
    move.l  Final_Screen_Address(pc),d0 ; Test par rapport au
    and.l   #$ffffff,d0         ; compteur vidéo pour savoir si
    move.l  $ffff8200.w,d1      ; l'overscan a bien été décleché
    lsl.w   #8,d1
    and.l   #$ffff00,d1
    sub.l   d1,d0
    cmp.l   #199*230,d0
    bpl.s   .Ok
    dbra    d2,.again2          ; on fait le test 2 fois pour
    subq.b  #1,mfp_Ada+3        ; être sûr. Sinon, on décrémente
    bne.s   .again              ; le délai.
    bra Adapt
.Ok subq.b  #1,mfp_Ada+3
    move.w  #$2700,sr
    rts

.TIMERB clr.b   $fffffa19.w     ; Intérruption locale de timer B qui
    clr.b   $fffffa1B.w         ; s'assure que seuls 199 lignes
    clr.l   $fffffa0a.w         ; d'overscan sont affichés.
.loop   move.b  (a1),d1         ; Attente de la fin de la ligne 199
    cmp.b   (a1),d1
    bne.s   .loop
    movep.l -6(a1),d1           ; sauvegarde addresse écran atteinte
    move.l  d1,Final_Screen_Address
    moveq   #-1,d1
    rte

TIMERB: clr.b   $fffffa19.w     ; Intérruption  TimerB
    clr.b   $fffffa1B.w         ; qui arrête l'intérruption
    clr.l   $fffffa0a.w         ; timer A d'overscan
    move.w  d6,$6C.w
RTE: rte

VBL: move   #$2700,sr           ; La Vbl qui initialise les
    move.b  #200-1,$fffffa21.w  ; registres et commence
    move.b  #8,$fffffa1B.w      ; l'overscan
    moveq   #0,d6
.sync cmp.b #$6E,$FFFF8209.w
    bmi.s   .sync
    move.l  #$920092,d0
    lea $ffff8209.w,a1
MFP: sub.b  (a1)+,d0            ; On se synchronise par
    lsr.w   d0,d0               ; rapport au compteur
    nop                         ; Vidéo
    move.b  d0,(a1)             ; Overcan gauche
    move.w  a1,(a1)
    move.b  d0,$fffffa19.w      ; 16
mfp_Ada move.b  #mfp,$fffffa1f.w ; 20
    move.w  #1,d0               ;  8
    lea $ffff8260.w,a0          ;  8
    move.w  a0,(a0)             ; Stabilisateur
    clr.b   (a0)
    move.b  d0,$fffffa19.w      ; 16
    swap    d0                  ;  4
    add.b   #230,d0             ;  8 une ligne overscan prend
    move.w  d0,a1               ;  4 230 octets: calcul de la
    swap    d0                  ;  4 prochaine adresse écran
    move.w  a1,d0               ;  4 pour se synchroniser la
    lea $ffff8209.w,a1          ;  8 prochaine fois
    move.w  a0,(a0)             ; Overcan droite
    clr.b   (a0)
    rte
```

4

Terminal forever <3



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 : S. Saurel
Nos experts techniques : S. Baring-Gould, R. Pinon, D. Lecan, N. Kassem, P-H. Gache, D. Duplan, A. Piroelle, F. Capon, M. Fontanier, S. Teodomante, G. Duval, J. Thiriet, P. Judicaël, S. Gombaud, S. Gombaud, M. Carol, T. Ranise, S. Berberat, P-A. Sunier, E. Mittelette, E. Deneuve, C. Villeneuve, C. Gigax, S. Abélard, T. Desmas, G. Leborgne, T. Ouvre, C. P. de Geyer, O. Philippot, E. Bataille, F. Dursus
Couverture : © 9comeback - **Maquette :** Pierre Sandré.
Publicité : PC Presse, Tél.: 01 74 70 16 30, Fax : 01 40 90 70 81 - 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 : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, février 2017

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 : 35 € (Monde Entier) souscription sur www.programmez.com

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

[2017]

DÉVELOPPEZ 10 FOIS PLUS VITE

WINDEV®

WINDEV 22 : ATELIER DE DÉVELOPPEMENT PROFESSIONNEL, COMPLET EN STANDARD

Gestion du cycle de vie complet: Idée, Conception, Développement, Génération, Déploiement, Exploitation • Un code multi-plateformes Windows, Linux, Java, Internet, Mobiles • Environnement ALM complet • Toutes les bases de données sont supportées, Big Data • Inclus: HFSQL, base de données locale, Client/Serveur, cluster, embarquée et cloud • Puissant RAD • Intégration continue • Générateur de fenêtres (UI) visuel & intuitif • Tableau de bord de vos applications • Audit statique & dynamique • Héritage et surcharge d'interface • Utilisation facile de charte graphique • Générateur de requêtes (QBE) visuel & intuitif • Tous les champs (contrôles) sont très puissants et livrés en standard: Champ de saisie, Tableau croisé dynamique (cube), Champ Planning, Champ Diagramme de Gantt, Champ Tableau de bord, Champ Table, Champ Graphe, etc • FAA: chaque application bénéficie automatiquement de Fonctionnalités Automatiques: export vers Excel, vers Word, envoi d'email, etc • Sécurité • Mot de passe de vos applications • Puissant générateur de rapports et codes-barres • Langage de 5ème génération: WLangage • Editeur de code intuitif avec puissant débogueur • Tests unitaires et tests automatiques • Versioning (GDS/SCM) • Webservices SOAP et Rest • Modélisation Merise et UML • .NET, 3-Tier, MVP • Support de tous les standards: XML, USB, Bluetooth, NFC, J2EE, OLE, ActiveX, RPC, SaaS, SMTP, FTP, OPC, DLNA, IoT, Sockets, API, Webservices... • Lien avec Lotus Notes, SAP, Google, Outlook • Multimédia, Domotique • Livré avec des centaines d'exemples et d'assistants • Génère le Dossier technique d'un clic • Télémétrie pour connaître l'utilisation réelle de vos applications • Support de 64 langues étrangères par applications • Gestion des suggestions et incidents utilisateurs • Support Technique Personnalisé Gratuit* • ...

CONSULTEZ PLUS DE 100 TÉMOIGNAGES DE PROFESSIONNELS SUR LE SITE PCSOFT.FR

Elu
«Langage le plus productif du marché»

VERSION EXPRESS GRATUITE
Téléchargez-la !

VU À LA TÉLÉ EN 2017
SUR **TF1**, SUR **BFMTV** ET SUR **M6**

Logiciel professionnel. * 15 requêtes gratuites sur la version en cours de commercialisation

Tél Paris: 01 48 01 48 88 *Tél Montpellier: 04 67 032 032*

Plus de 100 témoignages
Dossier complet gratuit

WWW.PCSOFT.FR