PROGRAMMEZ.COM

PROGRAMMEZ!

Mensuel n°185 - Mai 2015

le magazine du développeur

Quels futurs pour les langages de programmation ?

C++, C#, JavaScript, PHP, Swift
Les nouveaux modèles de programmation

Pourquoi les API sont partout ?

Créer, Déployer, Utiliser des API

Alan Turing

L'homme qui créa l'informatique moderne

Spartan

Le nouveau navigateur de Microsoft

Hacking

Sécurisez vos sites PHP !

Tobii EyeX

Traquez vos yeux

M 04319 - 185 - F: 5,95 € - RD

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

« C'est la faute à Turing »

Alan Turing vous dit vaguement quelque chose. Ceux qui s'intéressent à l'Histoire, savent qu'il fut un des casseurs de codes cryptés de l'Allemagne durant la 2e guerre mondiale. Mais son travail et sa vision dépassent très largement son rôle crucial pour casser les machines allemandes, dont la célèbre Enigma.

Il pose les bases de l'informatique « moderne » dès 1936 avec son article fondateur « on computable numbers, with an application to the entscheidungproblem ». Ce problème a été énoncé en 1928 par David Hilbert : une machine (mécanique) peut-elle dire si un énoncé mathématique est vrai ou faux ? Turing n'a que 24 ans quand cet article fondateur de l'informatique moderne est publié. Il imagine alors une machine « virtuelle », la machine universelle qui peut simuler une autre machine en fournissant un programme qui sera exécuté, en plus des données nécessaires. Le principe de l'ordinateur est là ! Bref, l'ordinateur hérite de la machine universelle de Turing. Cette idée fondamentale permettra au mathématicien de résoudre le problème de 1928 !

Turing est aussi le fondateur d'une autre science, tout aussi fondamentale : l'Intelligence Artificielle (IA). En 1950, il pose dès la 1ere phrase de son article « Computing machinery and intelligence » LA question : les machines peuvent-elles penser ? Cette question est au cœur de l'IA. Et son raisonnement devient : une machine peut-elle imiter l'homme ? C'est le test de Turing... Découvrez l'article dédié dans ce numéro.

Ca va piquer les yeux !

Ce mois-ci, un menu très très riche :

- *Je débute... avec les API*
- *Quels futurs pour nos langages de développement ?*
- *Le navigateur Spartan : vous savez le truc qui va tuer Internet Explorer*
- *Du wearable avec Android Wear*
- *JavaScript pour les Jedi*
- *Un peu de Bluetooth Low Energy*

Le thème du mois est le futur des langages de programmation. Il était important de revenir sur ce sujet brûlant pour les développeurs. Depuis un an, beaucoup de choses ont évolué : Java, SWIFT, JavaScript, C#. Mais l'autre élément est un changement dans la philosophie même du développement : notions de contrats, metaprogrammation, programmation fonctionnelle. Car finalement, au-delà du langage en lui-même, ce sont réellement les paradigmes de programmation qui changent peu à peu. Ces modifications apportent des changements profonds dans la structure du code, la qualité de celui-ci et sa robustesse, voire, aident à mieux maîtriser la prédictibilité du fonctionnement du code.

Et n'oubliez pas tous les témoignages de développeurs qui prouvent une nouvelle fois que le métier de développeur est réellement un super job !

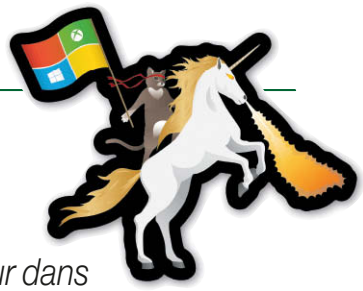
Bonne lecture

à lire dans le prochain numéro n°186 en kiosque le 30 Mai 2015

RÉSUMÉS
Retour sur les conférences développeurs Devoxx France 2015 et BUILD 2015

RESPONSIVE DESIGN
Une interface unique pour tous les écrans ? C'est la promesse du Responsive Design. Comment être Responsive ? Les avantages et inconvénients ? Les bonnes pratiques.

MakerFaire Paris 2015 :
les makers sont partout !



Spartan : le nouveau navigateur

Microsoft a annoncé il y a quelques mois l'existence du projet Spartan, son nouveau navigateur, qui remplacera à moyen terme Internet Explorer. Retour dans cet article sur ce projet : pourquoi est-ce que Microsoft a décidé de changer de stratégie, et qu'est-ce qui va changer pour nous, utilisateurs, développeurs et/ou designers.

Kévin Albrecht
et Sébastien Ollivier
Consultants Infinite Square
INFINITE SQUARE

La première version d'Internet Explorer apparaît dans les années 1990, pour concurrencer Nestcape. Au fil des années, le navigateur de Microsoft prend de plus en plus de parts de marché, pour atteindre près de 95% au début des années 2000. Depuis, ce chiffre ne fait que décroître, majoritairement au profit de ses nouveaux concurrents Firefox et Chrome, pour atteindre moins de 20% de nos jours Fig.1.
Côté mobile, Internet Explorer pâtit de l'arrivée tardive de Windows Phone sur le marché pour atteindre difficilement les 5% Fig.2. Ces chiffres sont la conséquence des arrivées de sérieux concurrents, notamment Firefox et Chrome comme cités précédemment, mais aussi la conséquence de choix qui se sont révélés négatifs pour la popularité d'Internet Explorer.

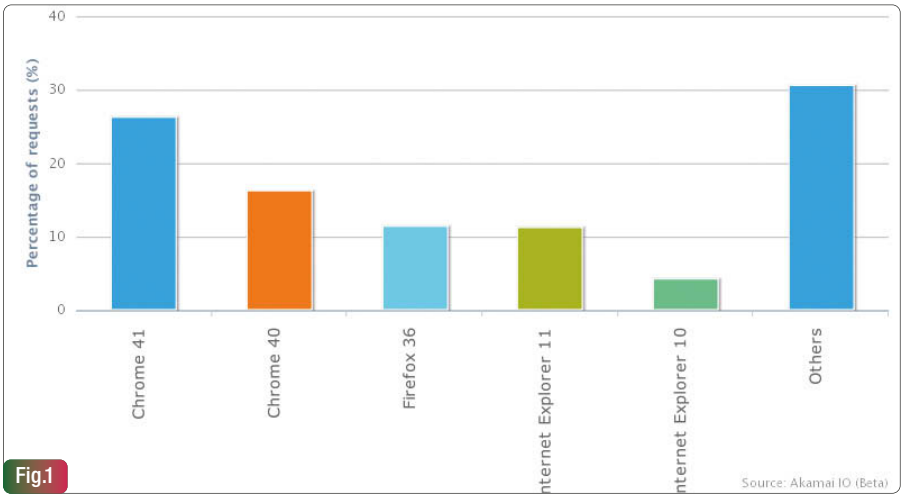


Fig.1

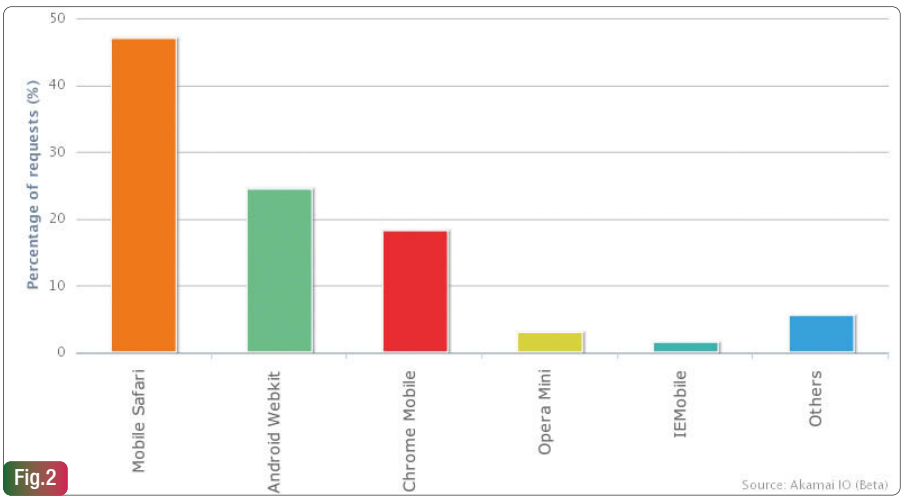


Fig.2

Ne pas respecter puis trop respecter les standards W3C

Le premier choix contestable concerne le respect des standards. Pendant de nombreuses années, et sur plusieurs versions, Microsoft a fait le choix de ne pas respecter tous les standards W3C et de proposer certaines implémentations propriétaires, au contraire des autres navigateurs concurrents. Ce choix est particulièrement visible sur la version 6 d'Internet Explorer qui comprend de nombreuses implémentations incomplètes ou défaillantes, notamment au niveau de CSS 2. Ce navigateur est considéré par les développeurs et designers comme celui nécessitant le plus de hack pour pouvoir afficher et exécuter correctement nos applications Web, ce qui a considérablement impacté négativement la popularité du navigateur. Pour rectifier le tir, Microsoft a ensuite décidé de respecter strictement les spécifications émises par le W3C. Bien que cela semble être une bonne idée, une implémentation trop stricte, par rapport aux concurrents comme Firefox et Chrome proposant une approche plus permissive, a donné l'impression d'une implémentation défaillante notamment à cause de la réputation des versions précédentes d'Internet Explorer.

Par exemple : le W3C spécifie que le fichier manifest, utilisé pour le cache HTML 5, doit être renvoyé avec le mime-type "text/cache-manifest". Alors que Chrome et Firefox ne vérifie pas le mime-type de ce fichier, Internet Explorer n'activera pas la mise en cache si le mime-type n'est pas correct. Pour un développeur n'ayant pas vérifié les spécifications W3C, ce comportement peut sembler être un bug d'Internet Explorer.

Versions majeures plutôt que mise à jour continue

Le second choix négatif est lié à la stratégie de mise à jour du navigateur. Alors que Firefox et Chrome misent sur de la mise à jour continue, le navigateur se mettant automatiquement à jour en arrière-plan pour garantir l'utilisation d'une version récente, Microsoft met à disposition plusieurs versions majeures et garantit un support pour ces versions (Internet Explorer en est à la version 11 alors que Chrome en est à la version 41). Microsoft doit ainsi s'assurer une rétrocompatibilité jusqu'à Internet Explorer 6 ce qui rend le développement des nouvelles versions plus long, plus fastidieux et moins efficace.
Pour repartir sur des bases saines et laisser derrière lui le passif négatif d'Internet Explorer, Microsoft a décidé de créer un nouveau navigateur, sous le nom de code Spartan, qui sera plus en adéquation avec le Web moderne et les contraintes actuelles.

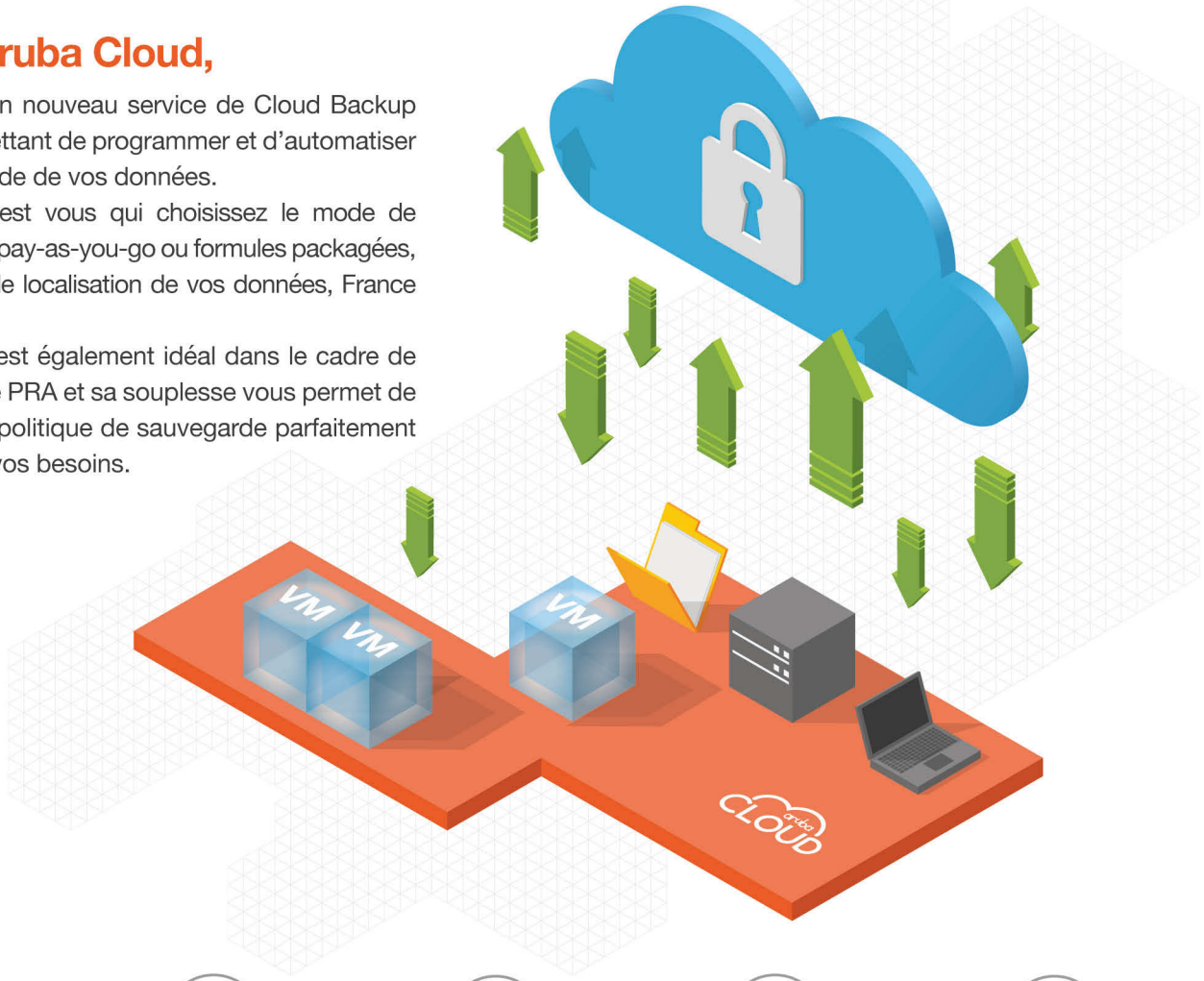
EdgeHTML : le nouveau moteur

Commençons par le moteur de rendu du nouveau navigateur Spartan. Il s'agit du moteur

Comment **automatiser** *mes sauvegardes* **sur le Cloud ?**

Avec Aruba Cloud,

profitez d'un nouveau service de Cloud Backup vous permettant de programmer et d'automatiser la sauvegarde de vos données.
De plus, c'est vous qui choisissez le mode de facturation, pay-as-you-go ou formules packagées, et le pays de localisation de vos données, France ou étranger!
Le service est également idéal dans le cadre de politique de PRA et sa souplesse vous permet de définir une politique de sauvegarde parfaitement adaptée à vos besoins.



Sécurité optimale

Simple d'utilisation

Sauvegarde automatique

Economique

Multi-plateforme

1 Quitte à choisir une infrastructure IaaS, autant prendre la plus performante!
Aruba Cloud est de nouveau **N°1 du classement des Cloud**
JDN / CloudScreener / Cedexis (avril 2015)

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

aruba CLOUD

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

MY COUNTRY. MY CLOUD.*

*MON PAYS, MON CLOUD.

EdgeHTML (ou tout simplement Edge) qui est un fork du moteur Trident, utilisé dans toutes les versions d'Internet Explorer depuis sa version 4.0. Il s'agit d'un moteur développé par Microsoft et spécialement conçu pour le Web moderne. Ce moteur sera disponible sur Windows 10, tout comme Internet Explorer 12, principalement pour des raisons de rétrocompatibilités. Si vous le souhaitez, il est déjà possible de tester le moteur EdgeHTML grâce aux Technical Previews de Windows 10 (versions plutôt stables de ce que nous avons pu tester), depuis la Build 9926. Pour cela, il est nécessaire d'activer le moteur de rendu en passant par la page "Experimental Features" (url about:flags dans Internet Explorer 11) puis en activant l'option "Enable Experimental Web Platform Features" Fig.3. Pour éviter d'hériter des handicaps d'Internet Explorer dès la sortie de son successeur, Microsoft a décidé de changer le user-agent de son navigateur pour se faire passer pour Chrome. L'idée derrière cela est d'éviter la pratique de "user agent sniffing" qui permet à un site d'identifier le navigateur de l'utilisateur afin d'activer un certain nombre de hacks visant à faire fonctionner le site correctement. Grâce à cela, Microsoft ne dévoile pas l'identité finale de son navigateur et lui permet de bénéficier d'un traitement classique. Le nouveau design de Spartan n'est pas disponible à l'heure où nous écrivons cet article mais son moteur peut d'ores et déjà être testé, comme nous l'avons vu précédemment. Microsoft insiste cependant sur le fait que Spartan sera le seul navigateur à disposer du nouveau moteur EdgeHTML final, les utilisateurs n'adoptant pas la version 10 de Windows ne pourront donc pas l'utiliser et resteront sur Internet Explorer.

Le Web moderne

Lorsque l'on parle de Web moderne, on entend ici que Spartan est conçu pour faire table rase de l'obsolescence d'Internet Explorer. Adieu les contrôles ActiveX, presque uniquement utilisés pour Internet Explorer, qui permettaient d'exécuter un programme via une page Web, ce qui était source de risques critiques. Adieu également les Browser Helper Objects, qui étaient un système de "plugins", mais qui étaient majoritairement utilisés pour installer des toolbars inutiles, voire malveillantes. Adieu enfin VBScript, permettant de piloter Internet Explorer en script. Spartan donnera la possibilité de créer des plugins directement développés en HTML, CSS et JavaScript, comme le font déjà ses principaux concurrents. Un point important a également été mis au niveau des performances. Voici un benchmark réalisé par AnandTech Fig.4. On y voit bien que le nouveau moteur se rapproche des performances de Chrome, voire même les dépasse sur certains points, laissant Internet Explorer loin derrière. Concernant le support HTML 5, les résultats n'ayant pas encore été dévoilés, on peut déjà observer une nette amélioration depuis la build 10041 de Windows 10 qui, même si elle n'était pas portée sur l'amélioration du moteur EdgeHTML, offrait déjà un joli score de 370/555 (ce qui montre une implémentation plutôt avancée des spécifications HTML 5 pour un moteur en cours de développement, dont la dernière version est une pre-release). A n'en pas douter, le support de HTML 5 et les performances seront nettement améliorés dans les prochaines builds. Microsoft s'investit dans l'implémentation des spécifications W3C et dans la conservation d'une certaine interopérabilité pour nous prouver que ce nouveau navigateur a bel et bien sa place dans le Web Moderne. Le nouveau moteur de Spartan promet plus de souplesse et de facilité pour les développeurs pour assurer un retour de Microsoft aux premières places sur le marché des navigateurs.

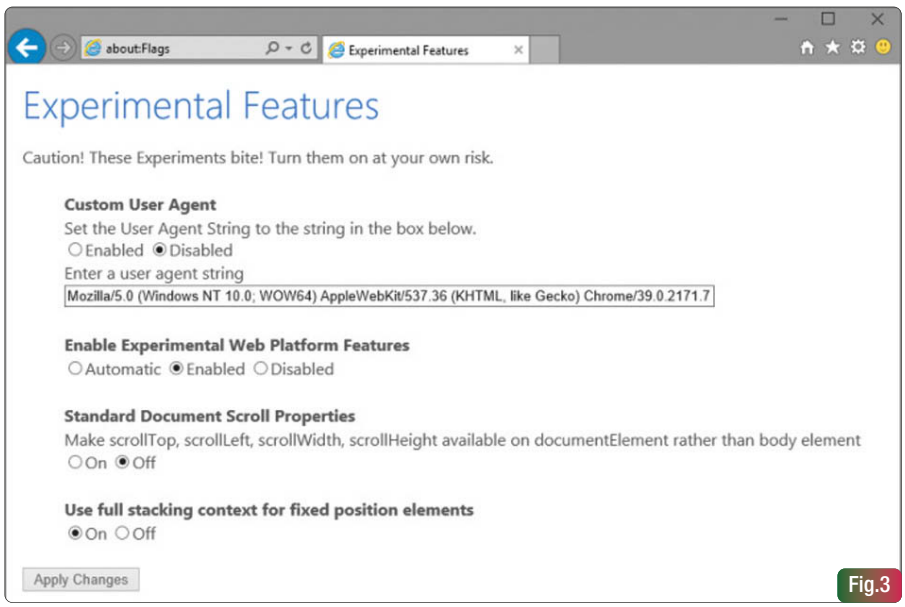


Fig.3

Fig.4

Browser Performance - Core i7-860

Benchmark	IE Old	IE Experimental	Chrome 40	Firefox 35	Percentage Change
Sunspider (lower is better)	149.7ms	**144.6ms**	260.9ms	220.1ms	3.4%
Octane 2.0 (higher is better)	9861	**17928**	17474	16508	81.8%
Kraken 1.1 (lower is better)	3781.2ms	**2077.5ms**	1992.8ms	1760.4ms	45.1%
WebXPRT (higher is better)	913	**1083**	1251	1345	18.6%
Oort Online (higher is better)	1990	**2170**	5370	3900	9%
HTML5Test (higher is better)	339	**344**	511	449	1.5%

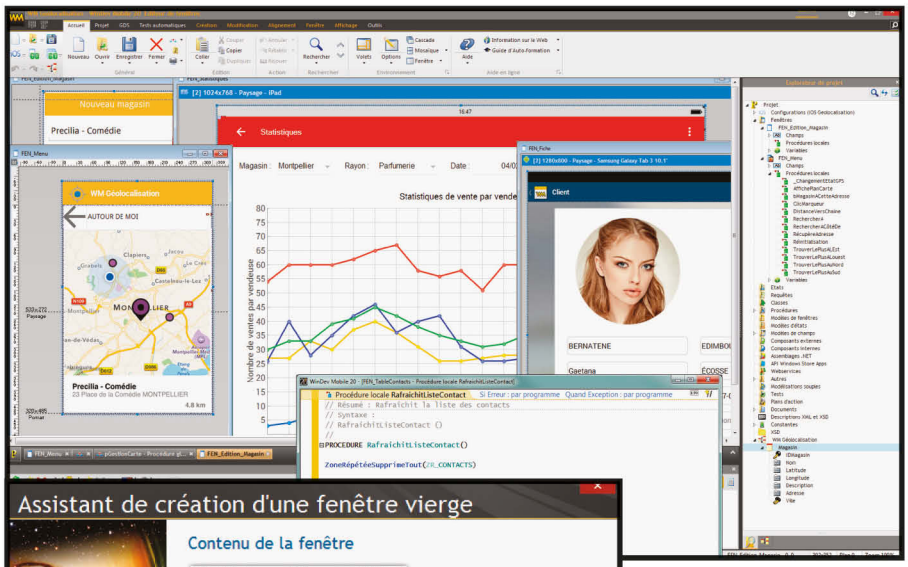
UI : une interface toute neuve

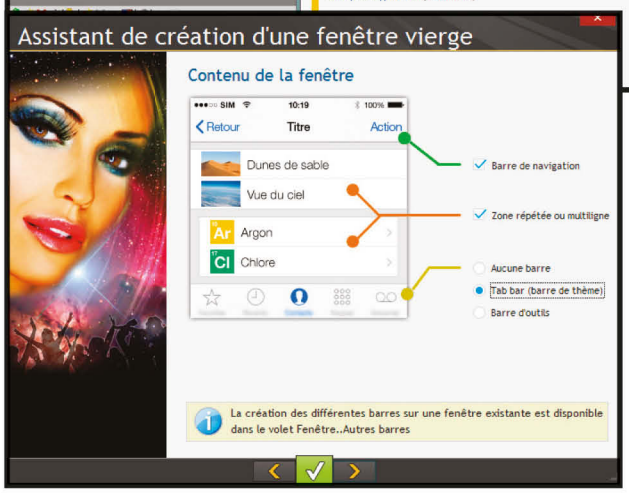
Maintenant que nous avons vu le nouveau moteur, le cœur du navigateur, voyons ce que Spartan propose de nouveau en termes d'ergonomie. Tout d'abord, les favoris et l'historique se retrouvent dans ce qui s'appelle le "hub", auxquels viennent s'ajouter la liste des téléchargements ainsi que la "reading list" qui représente une liste de pages Web que vous pouvez annoter comme étant à visiter plus tard. Spartan propose actuellement 4 nouvelles fonctionnalités majeures:

Cortana et la recherche intelligente

L'assistante Cortana est intégrée à Spartan. Cela veut dire qu'en fonction de vos recherches et de votre navigation, Cortana vous proposera un ensemble d'informations : la météo pour une

WINDEV MOBILE 20
LE DÉVELOPPEMENT NATIF
POUR TOUS LES MOBILES





Un environnement de développement complet, intégré, adapté au monde du «mobile»

PORTABILITÉ DE VOS APPLICATIONS

ANDROID, IOS, WINDOWS PHONE, WINDOWS MOBILE & CE

Avec WINDEV Mobile 20, une même application peut fonctionner sous les différents OS mobiles: iOS (iPhone, iPad), Android, Windows CE & Mobile, Windows Phone... Recompilez !

TOUS LES TYPES DE MOBILES

Développez pour tous les mobiles: téléphones, smartphones, pocket PC, terminaux, terminaux durcis, terminaux industriels, tablettes, netbook,...

CRÉEZ DES APPLICATIONS NATIVES POUR TOUS LES SYSTÈMES MOBILES

WINDEV Mobile 20 permet aux professionnels du développement de créer facilement des applications natives pour tous les mobiles: smartphones, tablettes et terminaux industriels. Et si vous possédez un existant WINDEV ou WEBDEV, vous pouvez le ré-utiliser.

UN ENVIRONNEMENT DE DÉVELOPPEMENT AUTONOME

Quels que soient le matériel cible et le système d'exploitation, la méthode de développement avec **WINDEV Mobile 20** est similaire.
L'environnement de développement est intégré, puissant, complet, intuitif, et il est adapté aux spécificités des mobiles.
Avec ou sans base de données, avec ou sans connexion au S.I. il n'a jamais été aussi facile de développer professionnellement sur mobile.

LE CYCLE DE VIE COMPLET EST GÉRÉ

WINDEV Mobile 20 est livré en standard avec tous les outils qui permettent de gérer le cycle de vie des applications: Générateur de fenêtres, Langage L5G, Débogueur, Générateur de rapports, Générateur d'installations, mais aussi Générateur d'analyses Merise et UML, Tableau de Bord du projet, Gestion-naire de Sources collaboratif, Générateur de dossier de programmation, Suivi des plannings,...

PROGRAMMEZ EN L5G: 90% DE CODE EN MOINS

Le langage de 5ème génération WLangage permet de développer plus vite qu'avec un langage traditionnel.
Ses fonctions évoluées rendent le code facile à écrire et à lire, facilitent à la fois le développement et la maintenance.

VERSION EXPRESS GRATUITE
Téléchargez-la !

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

Fournisseur Officiel de la Préparation Olympique

www.pcsoft.fr
Des centaines de témoignages sur le site

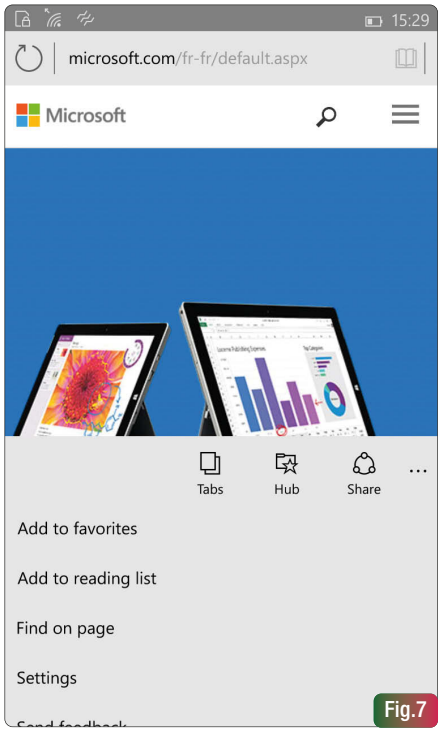


Fig.7

ville, la position sur Bing Map d'un restaurant, la traduction d'un terme surligné, etc. Pour pousser l'expérience encore plus loin, Cortana offrira également des informations contextuelles en fonction du site visité. Par exemple, si vous naviguez sur le site d'un restaurant, Cortana vous proposera son adresse, des avis, ses horaires d'ouvertures, son numéro de téléphone Fig.5.

La prise de note sur un site internet

Il sera également possible de prendre des notes directement dans Spartan, sur un site. Vous pourrez ainsi dessiner, surligner des mots, prendre des notes directement sur un site puis envoyer vos notes par mail ou les importer dans OneNote Fig.6.

La reading view

Option déjà existante sur Safari, Spartan proposera un mode Reading View. Ce mode permet de n'afficher que les informations importantes d'une page, pour permettre une lecture facilitée en se concentrant sur l'essentiel.

Spartan pour mobile

Coté mobile on peut d'ores et déjà tester le nouveau navigateur Spartan sur Windows Phone depuis la dernière build 10051 de début Avril. Cette version mobile n'est aucunement différente de la version bureau d'un point de vue du support des standards Web, mais inclut déjà le nouveau design orienté mobile. S'agissant d'une pre-release, il est fort probable que ce design soit amené à évoluer Fig.7.

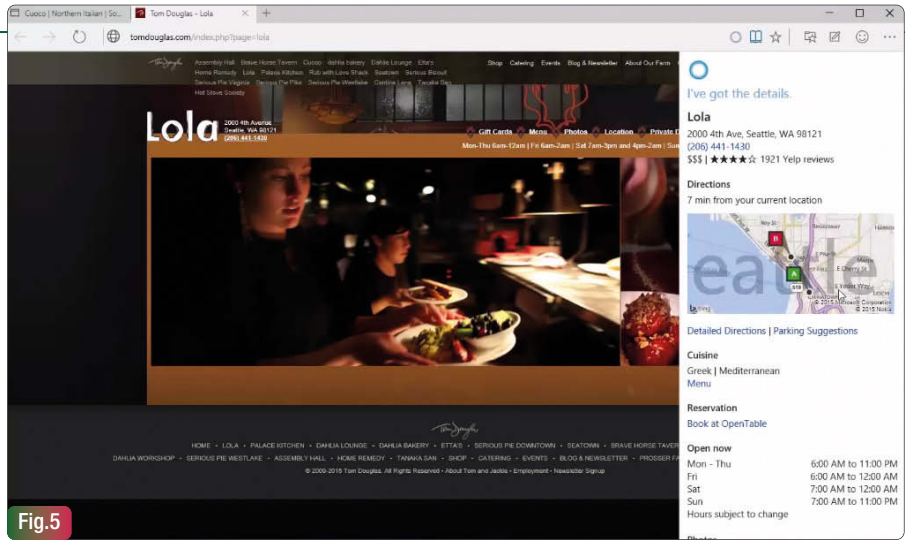


Fig.5

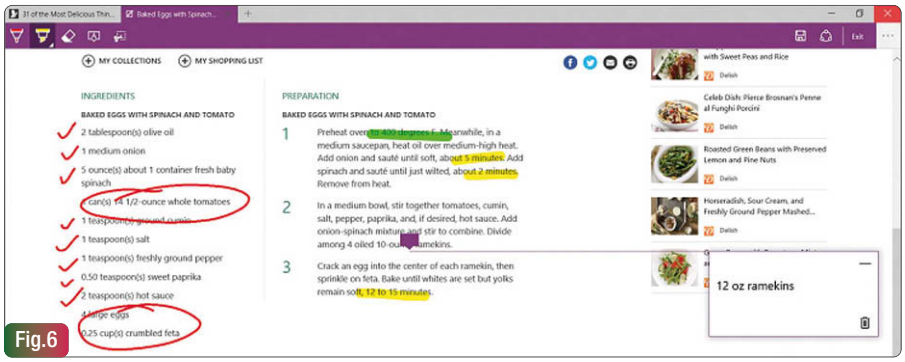


Fig.6

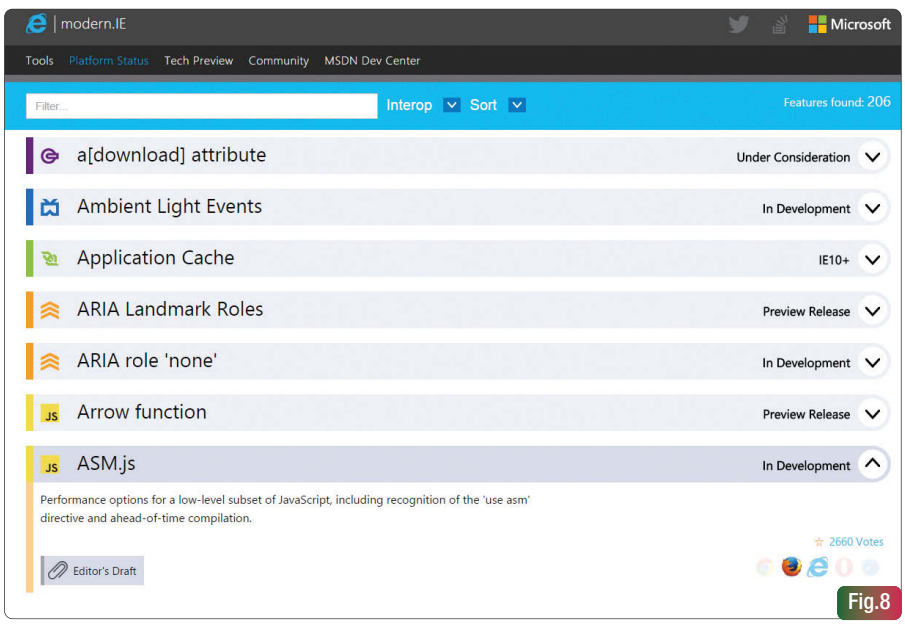


Fig.8

La communauté

Soucieux de proposer un navigateur performant et adapté aux utilisateurs, Microsoft a mis en place une boîte à suggestions pour Spartan que vous pouvez retrouver à l'adresse suivante: http://goo.gl/nZ6FXD

Cette plateforme aide Microsoft à suivre les retours des testeurs et à collecter les suggestions des utilisateurs pour fournir un navigateur encore plus adapté. Microsoft propose également un site Web sur lequel on peut suivre l'avancement des fonctionnalités implémentées par Spartan, comme par exemple les nouvelles briques introduites par EcmaScript 6 ou encore HTTP/2 Fig.8.

A travers cet article, nous avons vu que Microsoft souhaite se replacer dans le domaine des navigateurs en proposant un navigateur tout neuf. Nous vous conseillons de tester les premières versions de Spartan, qui s'avère déjà très prometteur et qui pourrait définitivement effacer Internet Explorer des mémoires des développeurs, designers et utilisateurs.

Tirer parti du meilleur de Cisco grâce à PRTG Network Monitor

Analyser les données

Superviser le trafic réseau avec NetFlow

NetFlow fournit des informations très détaillées sur le trafic réseau. PRTG se concentre sur l'analyse de celles qui revêtent une importance critique – Top Talkers, Top Connections ou les Top Protocols – afin d'envoyer des alertes si nécessaire. Les résultats, représentés sous forme de graphiques, offrent une vue d'ensemble sur le trafic des données au sein du réseau.

PRTG embarque également des capteurs NetFlow personnalisables qui délivrent les valeurs les plus pertinentes pour son utilisateur. Le logiciel peut en outre extraire des données à l'aide des protocoles xFlow ou du reniflage de paquets.

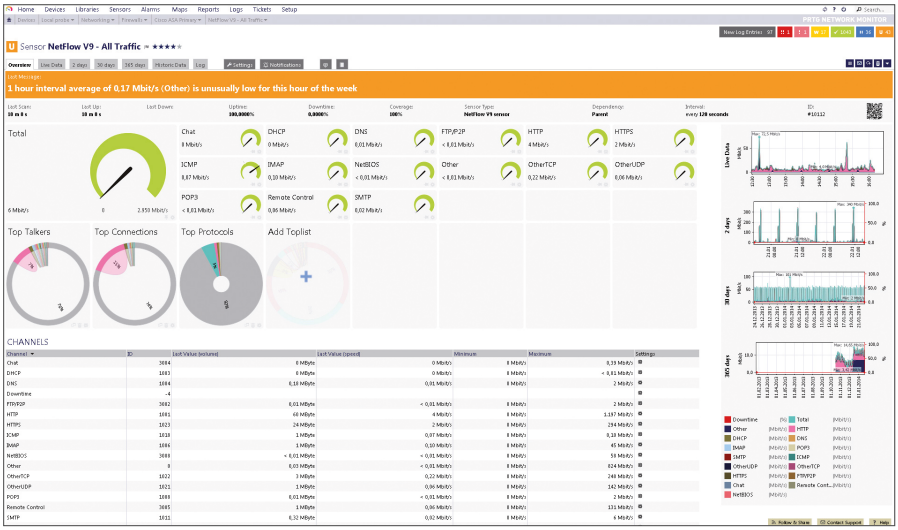
Superviser le trafic réseau avec SNMP

Nombre de périphériques Cisco ne sont pas compatibles avec NetFlow. Elles doivent néanmoins être supervisées afin de déterminer l'état du périphérique et du trafic réseau, et garder un aperçu de l'ensemble du réseau. SNMP est compatible avec les commutateurs, les routeurs, mais aussi les serveurs, logiciels, imprimantes et même certaines machines à café.

SNMP fournit bien moins d'informations que NetFlow et mobilise également moins de ressources réseau. Il peut être judicieux d'activer SNMP sur les périphériques qui fonctionnent également avec NetFlow. Comme avec ce dernier, les données SNMP sont générées et traitées par chacun des périphériques. Si les capteurs PRTG préconfigurés ne sont pas assez précis, des requêtes spécifiques peuvent être définies pour chaque périphérique, à l'aide de fichiers MIB. L'administrateur peut programmer des seuils critiques au-delà desquels il reçoit des alertes et alarmes, et configurer les relations et dépendances vis-à-vis des autres capteurs.

Optimiser la qualité des communications téléphoniques via IP SLA

Les périphériques Cisco compatibles IP SLA mesurent la qualité des données en streaming, comme les VoIP ou les conversations vidéo reposant sur divers protocoles.



Aperçu d'un Netflow Cisco 9 capteurs

PRTG interprète les données IP SLA, les convertit en graphiques et avise l'administrateur d'éventuelles anomalies. Les capteurs QoS et CBQoS intégrés à PRTG s'acquitteront des requêtes plus complexes ou des périphériques incompatibles avec le protocole IP SLA.

Comment exploiter les données

Alertes et alarmes

Les seuils critiques définis envoient des alertes par e-mail, SMS, trappe SNMP et par notifications Push (bêta). Ainsi, l'administrateur est assuré que tous les périphériques Cisco fonctionnent normalement et que la qualité des appels téléphoniques est irréprochable.

Visualiser et publier

Les données présentées sous forme de graphiques, fournissent une vue d'ensemble de l'environnement réseau. En outre, certaines valeurs sont représentées sous forme de tableau pour une analyse plus poussée.

Le logiciel propose aussi des options pour intégrer les résultats dans des cartes HTML en temps réel ou des rapports transmis à intervalles programmés.

Au-delà des périphériques Cisco : tout votre réseau en un coup d'œil

PRTG permet de superviser environnements virtuels, applications, bases de données, systèmes de stockage, trafic e-mail et bien plus encore. Associée à un dispositif de surveillance exhaustive des périphériques Cisco, la solution de surveillance PRTG permet aux administrateurs d'accéder en quelques clics à un véritable arsenal d'outils pertinents et faciles d'utilisation pour assurer le bon fonctionnement du réseau et se prémunir contre les mauvaises surprises.

EN SAVOIR PLUS:
www.paessler.fr/prtg-cisco

Pour plus de renseignements sur PRTG Network Monitor, contactez :

Christophe da Fonseca
T : 06 59 77 88 56
info@paessler.com
www.paessler.fr

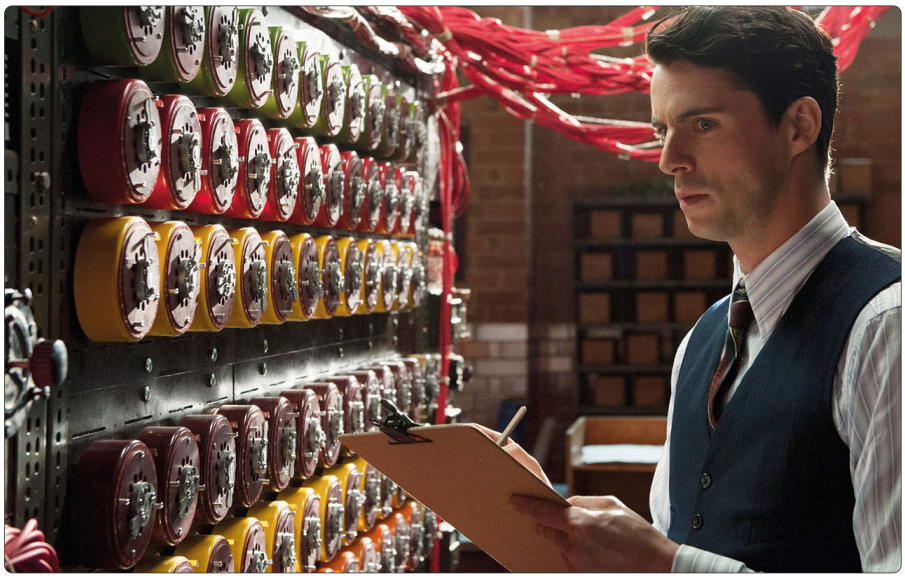
PAESSLER
the network monitoring company

538024/FR/20150416

(Les) héritiers d'Alan Turing !

Certains Hommes changent une vie, puis d'autres vous changent le Monde radicalement, sans compromis, sans retour, sans attente, juste parce qu'il fallait le faire, juste parce qu'ils devaient le faire pour l'intérêt et le bien-être de tous !

Gregory Renard
xBrain



Alan Mathison Turing, mathématicien de génie, cryptologue et informaticien, né le 12 juin 1912 dans le Grand Londres fait clairement partie de cette catégorie d'Hommes qui changent le Monde avec Force et Conviction.
Inconnu et incompris par ses contemporains, il est l'un des pionniers et pères fondateurs des sciences de l'informatique ainsi que de l'intelligence artificielle. Ayant permis de raccourcir de près de 2 ans la Seconde Guerre mondiale, il a épargné des dizaines de millions de vies, contribué à ramener la paix au sein de notre civilisation alors trop absurde pour le reconnaître et le remercier de son vivant pour ses actes de bravoure ! Sa réhabilitation et les excuses officielles du gouvernement britannique en 2009 ne pourront effacer les souffrances physiques, physiologiques et psychologiques infligées au père de l'informatique ! Ses travaux ont radicalement impacté notre société contemporaine, notre vie, votre vie quotidienne. Que vous soyez Geek ou non, il ne se passe pas une journée sans que vous ne soyez impacté par les fondements établis par Alan Turing il y a que quelques décennies ! Alan Turing est un effet papillon positif à lui tout seul dans l'espace et dans le temps dont nous découvrirons encore longtemps après les effets domino !

La Machine de Turing et la calculabilité

C'est en 1936 que Turing pose les bases de l'informatique moderne avec son article « On Computable Numbers, with an Application to the Entscheidungsproblem » répondant à un problème posé par le mathématicien Hilbert dans les théories axiomatiques et plus précisément au problème de la décision (Entscheidungsproblem) : est-il possible de trouver une méthode « effectivement calculable » pour décider si une proposition est démontrable ou non. C'est ainsi que Turing imagine la fameuse machine de Turing. Forme d'automate abstrait qui formera la base de la théorie des automates et de la calculabilité (cfr le problème de Hilbert). Une forme de « machine universelle » pouvant accomplir les tâches de n'importe quelle autre machine simple. La machine de Turing doit pouvoir effectuer un calcul complexe par le séquençage d'opérations simples, tout étant une notion de perspective et de contexte unitaire. Elle est composée de 4 éléments :

- Un ruban de papier illimité composé de cases successives,
- Une tête de lecture/écriture afin de lire le contenu de chaque case,
- Un registre mémorisant l'état de la machine (son contexte),
- Une table d'actions ou programme à appliquer par la tête de lecture.

Nous retrouvons ainsi toutes les composantes de nos ordinateurs contemporains, un ruban illimité pour mémoire et le temps de calcul pour le nombre d'opérations à effectuer correspondant à nos processeurs. La table ou programme déterminera le mouvement de cases du ruban vers la droite ou vers la gauche ainsi que l'opération de lecture ou écriture à opérer sur celles-ci. Cette machine, pouvant ainsi calculer comme un être humain discipliné, ne pourra réaliser que la seule opération pour laquelle elle a été programmée. Turing projette alors une machine universelle capable de simuler toute autre machine simple en lui fournissant uniquement des programmes codes à exécuter avec une série de données à manipuler. La machine de Turing devient ainsi le modèle abstrait de l'ordinateur contemporain en formalisant le concept d'algorithme et de langage informatique.

Christopher, la Bombe de Turing !

Les actes de courage ou bravoure ne se passent pas qu'au cœur des champs de batailles, ils peuvent aussi se passer au sein des laboratoires.
C'est ainsi qu'en pleine seconde guerre mondiale, Alan Turing s'attaque avec un petit groupe d'experts en cryptographie au défi impossible de « hacker » la machine Enigma alors réputée inviolable.
Cette machine ou plutôt famille de machines électromécaniques portables permettaient le chiffrement et déchiffrement de l'information ! Inventée par l'Allemand Arthur Scherbius sur base d'un brevet de Hugo Koch, elle sera utilisée par les Allemands afin de sécuriser les communications entre leurs différentes divisions militaires durant la Seconde Guerre mondiale. Je vous invite à parcourir la page Wikipédia de la machine Enigma expliquant en détails son fonctionnement, votre côté Geek-Mathématicien se délectera des informations ainsi que de l'ingéniosité du système... Pas moins de 1,59 x 10 exposant 20 combinaisons possibles pour tenter de décrypter un message encodé avec Enigma, et ce, de manière quotidienne ! Autant rechercher une aiguille dans une grange remplie de bottes de foin ! Mais c'était sans compter sur l'équipe ingénieuse de Bletchley Park pilotée par Alan Turing qui, en capitalisant sur le manque d'imagination et sur les erreurs humaines du

Offre spéciale abonnement !

et aussi...

PROGRAMMEZ!
le magazine du développeur

devolo
The Network Innovation

Pour un abonnement 2 ans, Devolo et Programmez! vous offrent un kit complet CPL dLAN 550
d'une valeur de 79,90 €

Pour seulement 94,90€

(au lieu de : 158,90 €, abonnement 2 ans / 22 numéros : 79 € + 15,90 € de frais logistiques et postaux)

Attention :
- *cette offre est strictement limitée à la France Métropolitaine et à la Corse.*
- *quantité limitée, jusqu'à épuisement des stocks. cette offre est susceptible de s'arrêter à tout moment.*

2 ans 22 numéros
94,90€

2 ans 22 numéros
79€
seulement (*)

1 an 11 numéros
49€
seulement (*)

Spécial étudiant
39€(*)
1 an 11 numéros

(*) Tarifs France métropolitaine

ABONNEZ-VOUS !

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

☐ ***Abonnement 1 an au magazine :*** 49 € (au lieu de 65,45 €, prix au numéro)

☐ ***Abonnement 2 ans au magazine :*** 94,90 € (au lieu de 158,90 €, kit CPL dLAN offert)

☐ ***Abonnement spécial étudiant 1 an au magazine :*** 39 €
Photocopie de la carte d'étudiant à joindre

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

☐ M. ☐ Mme ☐ Mlle Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

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

E-mail : @

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

☐ Je souhaite régler à réception de facture

** Tarifs France métropolitaine*

PROG 185
***Offre limitée**, valable jusqu'au 30 mai 2015*

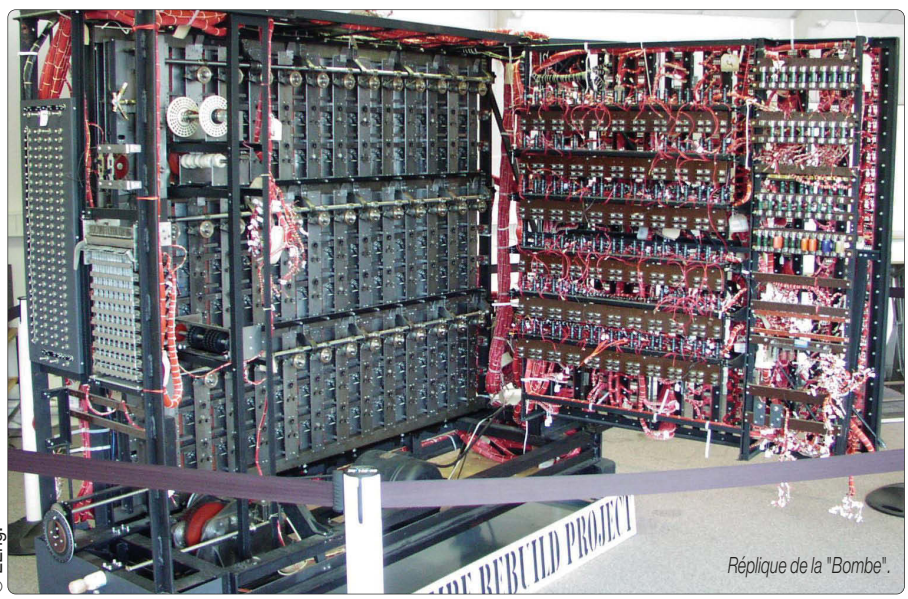
commandement allemand, arrivera à diminuer la complexité combinatoire et ainsi à craquer le code pour déchiffrer les messages cryptés par l'armée Allemande.
Effectivement, Turing partait entre-autres de la probabilité et fréquence de types de messages ou d'utilisation des mots par les Allemands comme « météo », « munitions », « chiffres », ...
Observez l'exemple repris de Wikipedia, cette phrase en clair
« ATTAQUECESOIRSURWIKIPEDIA », une fois chiffrée, donnera
« YAOPWMKLRBFZLVCXKTROTQALD » Fig.A.
Vous retrouvez ainsi en rouge et gris les paires successives T=>A, A=>P et enfin P=>T faisant apparaître des boucles dans le chiffrage des messages. La mise en parallèle des phrases chiffrées et non chiffrées permet à Turing de comprendre la notion de boucle et d'optimiser les tests de configurations. La mise en contexte et l'utilisation de mots réguliers par les Allemands permettra ainsi a Turing et son équipe de réduire de manière drastique le nombre de combinaisons et ainsi de passer à 26*26*26*60 = 1 054 560 combinaisons possibles ! Le terme étant toujours d'actualité, Turing utilisera donc la stratégie de bombe afin d'ébranler et déchiffrer le code allemand. Mais pas si simple que cela et pour cause le nombre de combinaisons reste encore en dehors de la capacité de traitement à la main par plusieurs cerveaux humains !
Mais pas pour « Christopher » ou Bombe de Turing, le premier ordinateur, il permettra de décrypter de manière industrielle les messages Allemands (nom de code ULTRA), et de déjouer les attaques ou mouvements allemands en vue de remporter la Seconde Guerre mondiale.
« Christopher » étant le nom donné par Turing à sa machine en mémoire de son défunt camarade. Qui sait ce qui se serait passé sans cette invention incroyable, pièce maîtresse de notre histoire et de nos libertels préservés.
Nous pourrions ainsi dire, comme si bien présenté dans le film « Imitation Game », qu'il s'agissait de « Contexte », qui une fois établi, fait chuter le nombre de combinaisons de manière drastique pour permettre à la bombe de Turing d'effectuer le travail de milliers d'humains dans leur propre intérêt.
Peut-être devons-nous y voir un signe d'encouragement face aux détracteurs et créateurs d'alarmes face à l'avènement des disciplines de l'intelligence artificielle ?

Fig.A

A	T	T	A	Q	U	E	C	E	S	O	I	R	S	U	R	W	I	K	I	P	E	D	I	A
Y	A	O	P	W	M	K	L	R	B	F	Z	L	V	C	X	K	T	R	O	T	Q	A	L	D

En route vers le Test de Turing et l'Intelligence Artificielle !

La guerre terminée, Alan Turing travaillera pour le NPL (National Physical Laboratory) où, inspiré par les travaux de Von Neumann, il rédigera le premier projet détaillé d'un ordinateur appelé Automatic Computing Engine (ACE). Pour différentes contraintes et incompatibilités de personnes, il poursuivra ses travaux à Cambridge avec Max Newman sur l'un des tous premiers ordinateurs, le Manchester Mark I (Ferranti) pour lequel il sera en charge de sa programmation.
Les années passent et en parallèle de ses travaux, Turing continue ses réflexions sur la réunion de la science et de la philosophie. En octobre 1950, il rédige l'article « Computing Machinery and Intelligence » où il explore le problème de l'Intelligence Artificielle et propose une expérience maintenant mieux connue sous le nom de « Test de Turing » afin de pouvoir à terme qualifier une machine de « consciente ».



Réplique de la "Bombe".
© EEng.

Pour rappel, le test de Turing consiste à mettre en confrontation verbale un humain avec un ordinateur et un autre humain à l'aveugle. Si l'homme qui engage les conversations n'est pas capable de dire lequel de ses interlocuteurs est un ordinateur, on peut considérer que le logiciel de l'ordinateur a passé le test avec succès. En 1966, le programme ELIZA de Joe Weisenbaum en SNOBOL sera le premier programme à donner l'illusion de répondre au Test de Turing en reformulant des affirmations de l'humain en questions que l'ordinateur lui pose ensuite. Le point le plus intéressant avec le Test de Turing est que ce test ne détermine pas directement si l'ordinateur se comporte de façon intelligente, mais si l'ordinateur se comporte comme un être humain. Effectivement, l'ordinateur devra se trouver dans la moyenne de l'intelligence d'un Être humain, sans être trop en dessous ou trop au-dessus !!!

Conclusion

À la lecture de la vie d'un génie comme Alan Turing, nous pouvons être en droit de nous poser la question si notre société est bien capable d'accueillir et comprendre les travaux de ce type d'envergure. Je ne le crois pas malheureusement, y compris encore aujourd'hui ! Force est de constater par les souffrances d'Alan Turing relatives à son homosexualité et la non-reconnaissance de ses actes de bravoure pour raisons de secret défense, que nous avons encore beaucoup de chemin à parcourir afin de permettre à des génies de l'envergure d'Alan Turing de s'épanouir dans notre société, cette communauté pour laquelle ils se battent tant !
Heureusement, le trend Geek de notre société par la présence technologique exponentielle facilite la vie de ces passionnés d'informatique, d'algorithme ou de mathématiques. Ces passionnés, dignes héritiers de Turing, que sont les Geeks.

MERCI Monsieur Turing pour votre génie,
MERCI pour votre travail et votre partage au travers de vos publications.
Alan Turing, le premier Geek communautaire dont nous sommes tous les héritiers ! ⊞

Sources et Liens :

http://fr.wikipedia.org/wiki/Machine_de_Turing
http://fr.wikipedia.org/wiki/Enigma_(machine)
http://fr.wikipedia.org/wiki/Cryptanalyse_d%27Enigma
https://www.youtube.com/watch?v=QZFol3gH1pg
http://fr.wikipedia.org/wiki/Test_de_Turing

Comment sécuriser vos applications Web avec MkFramework ?

Vous avez pu découvrir dans PROGRAMMEZ! (n° 167, 170 ,173 et 180), MkFramework, un autre framework. Dans ce tutoriel, je vous propose d'apprendre à sécuriser vos applications Web avec celui-ci.

Michaël Bertocchi
@dupot_org

Aujourd'hui beaucoup trop de développeurs Web semblent ignorer que nos sites Web font l'objet de nombreuses attaques, pourtant elles sont bel et bien nombreuses et peuvent être évitées plus ou moins simplement. Je vais vous décrire ici la majorité d'entre elles et les moyens pour vous en prémunir au mieux.
Je travaille depuis des années avec des sociétés d'audit de sécurité qui vérifient nos applications Web et nous indiquent les failles et les moyens de contournement de celles-ci. Au fil de cette expérience, j'ai implémenté ces conseils au sein du mkframework.

La sécurisation du serveur

Ne divulguer aucune information pouvant être exploitée et faciliter la découverte de faille.
Avec le paramétrage par défaut, votre serveur Apache retourne beaucoup d'informations sur la version de votre OS, de votre serveur Apache...
Commencez par passer la directive "ServerTokens" à "prod" dans votre fichier security.conf (de votre Apache).
Même chose pour les erreurs php, passer la variable "display_errors" à "off" dans le fichier php.ini
N'autorisez pas votre site à être imbriqué dans un autre, sinon un hackeur pourrait imbriquer celui-ci dans une iframe et ainsi faire du phishing.
Pour cela ajoutez la directive "Header set X-Frame-Options: "sameorigin"" dans votre fichier security.conf (de votre Apache).

La sécurisation de votre application Web

Le déni de service ou DDoS

Il existe deux types de DDoS: le technique et l'applicatif. Le premier consiste à faire "tomber" un serveur, le second consiste à rendre la plateforme inutilisable.
Imaginez un gestionnaire de mail bloquant les comptes au bout de la troisième tentative, un simple script faisant des tentatives infructueuses en boucle bloquerait un nombre important de comptes.
Pour éviter cela vous avez plusieurs solutions :

- Ne pas bloquer définitivement le compte mais pendant un temps déterminé
- Alerter par email au bout d'un certain nombre de comptes bloqués sur une courte période.

L'injection SQL

L'idée ici est d'envoyer une chaîne de caractères permettant de modifier la requête SQL initiale. Pour s'en prémunir, on conseille d'utiliser les procédures stockées (dans le SGBD) ou les "prepare statements" de PDO.
Le mkframework, comme beaucoup d'autres frameworks/ORM utilisent cette fonctionnalité: pour cela utilisez les méthodes findMany() et findOne() du framework.

Par exemple :

```
$this->findMany('SELECT USER_PKEY FROM users WHERE USER_Login= ? AND USER_Password= ?',$login,$password) ;
```

Les paramètres passés ici le sont indépendamment de la requête, PDO se charge de les sécuriser.

Le XSS ou la modification de code via une variable

Les variables d'environnement: $_GET et $_POST peuvent contenir des caractères permettant de modifier le site affiché, pour s'en prémunir, le framework convertit les caractères pouvant poser problèmes (chevrons, apostrophes, null byte...)
Pour en bénéficier, utilisez la méthode _root::getParam() plutôt que les tableaux d'environnements PHP.

Le XSRF ou la soumission d'un formulaire via un site tiers

Imaginez que l'on puisse, en naviguant sur un site piège soumettre un ordre de virement de votre banque. Pour s'en prémunir, le framework vous propose un plugin générant un "jeton" unique et éphémère. Utilisez donc le plugin plugin_xsrf en 5 étapes :

1 Dans la page affichant le formulaire, on génère le jeton, puis on l'assigne à la vue

```
Public function _edit(){

    //récupération du tableau des messages d'erreurs
    $tMessage=$this->processSave();

    //récupération de l'objet auteur
    $oAuteur=model_auteur::getInstance()->findById( _root::getParam('id') );

    //création d'un objet vue
    $oView=new _view('auteur::edit');
    //assignation de l'objet auteur
    $oView->oAuteur=$oAuteur;

    //création du jeton
    $oPluginXsrf=new plugin_xsrf();
    //assignation du jeton à la vue
    $oView->token=$oPluginXsrf->getToken();

    //assignation de la vue au layout
    $this->oLayout->add('main',$oView);

}
```

2 Dans la vue incluant le formulaire, on affiche le jeton dans un champ caché.

```php
<?php
//création d'un plugin formulaire avec l'objet issu de la base de données
$oForm=new plugin_form($this->oObject);
//on lui passe le tableau des messages d'erreur
$oForm->setMessage($this->tMessage);
?>
<form action="" method="POST" >
(…)


<?php //on affiche le jeton
echo $oForm->getToken('token',$this->token)?>

</form>
```

3 Lors du traitement de données, on vérifie ce jeton et on l'invalide

```php
private function processSave(){
  if(!_root::getRequest()->isPost() ){ //si ce n'est pas une requête POST on ne soumet pas
    return null;
  }

  $oPluginXsrf=new plugin_xsrf();
  //on vérifie que le jeton est valide
  if(!$oPluginXsrf->checkToken( _root::getParam('token') ) ){
    return array('token'=>$oPluginXsrf->getMessage() );
  }
  (...)
}
```

4 On peut renforcer son efficacité en activant la gestion en session dans le fichier conf/site.ini.php du projet

```
[security]
xsrf.session.enabled=1
```

5 Rendez le sel unique: pour éviter de rendre ce jeton prédictible, modifiez le sel par défaut de votre application en modifiant le plugin plugin_xsrf

```php
public function plugin_xsrf(){
  $this->sSalt='fdsfhoyu679hjfdsAfef';
  (...)
}
```

Le vol de cookie de session

Les cookies pouvant être accédés à la fois par Javascript et langage serveur (php), des personnes mal intentionnées peuvent via divers moyen parvenir à récupérer certains cookies de session sur votre ordinateur (par le biais d'email par exemple). Pour vous en protéger, il suffit de modifier certains paramètres de votre application, dans le fichier conf/site.ini.php

```
[auth]
;note : >= php5.2 dans le php.ini
session.use_cookies = 1
session.use_only_cookies = 1
session.cookie_httponly=1
```

Le débordement de session

L'action consiste ici à éditer/modifier un élément qui ne nous appartient pas : par exemple lire un email qui n'est pas le notre, annuler une commande d'un autre client...
Pour cela, il faut à chaque page/action vérifier que le contenu affiché/modifié appartient bien au compte connecté.
Vous pouvez utiliser la fonction getAccount() pour récuperer le compte connecté :

```
_root::getAuth()->getAccount()->cle_primaire
```

Armé de sa clé primaire, vous pourrez vérifier le lien de filiation du contenu.

La racine Web, ou Web root

N'importe qui peut taper dans l'url de son navigateur ../ et ainsi naviguer dans l'arborescence du site visité : pour éviter cela on restreint la navigation à une partie publique du site.
Vous pouvez par exemple paramétrer ainsi votre virtualhost :

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
  ServerName www.mon-site.com
  DocumentRoot /var/www/mkframework/data/genere/monApplication/public
  <Directory />
    Options FollowSymLinks
    AllowOverride None
  </Directory>
  <Directory /var/www/mkframework/data/genere/monApplication/public>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride All
    Order allow,deny
    allow from all
  </Directory>
</VirtualHost>
```

Le vol de login, simplifiant la brute force

Tous les endroits où vos internautes se connectent ne sont pas forcément sécurisés, pensez à rendre les champs d'identification non enregistrables. Pour cela, dans vos champs ajouter l'attribut html autocomplete="off"

Conclusion

Vous avez pu lire ici plusieurs conseils/bonnes pratiques pour sécuriser vos applications Web. J'espère que ceux-ci vous seront utiles et vous permettront de mieux vous prémunir de ces failles trop généralisées.
Un dernier conseil que l'on donne souvent : ne faites pas confiance aux données provenant de l'extérieur de votre application. Ne pensez pas à tort que parce que le formulaire limite sa saisie (par un menu déroulant ou que l'id est passée dans un champ caché) que sa valeur est sûre : un simple appui sur la touche F12 permet de modifier un formulaire, alors imaginez les outils que les hackeurs ont à leur disposition pour les modifier à leur convenance.
Si vous développez dans un cadre professionnel des applications "publiques" pensez à sensibiliser vos responsables sur l'importance de faire passer des audits de sécurité à ces applications plus ou moins critiques.

Développeur « Le plus beau métier du monde.»

Avec mes 26 ans de carrière dans le domaine du développement, on me pose souvent la question : "Est-ce que le métier de développeur a évolué depuis tes débuts ?"

Eric Vernié
Créateur de l'association
Fier d'être développeur
http://fierdetredeveloppeur.org/

C'est toujours une question à laquelle il est difficile de répondre objectivement, car chacun a sans doute sa propre expérience et son propre contexte à exposer, c'est pour cela que je tiens à préciser que ses quelques lignes n'engagent que moi et ne peuvent être que le reflet de ce que j'ai vécu durant ma carrière. Elles peuvent sans doute être complétées par votre propre expérience, et vous avez tout à fait le droit de ne pas être en phase et d'accord avec moi.
Cependant, je répondrai simplement oui, à la question, "est-ce que ce métier a évolué depuis tes débuts ?"
Est-ce que le développeur a évolué ? La réponse est tout aussi simple, oui également et heureusement, mais est-ce que l'âme, l'esprit d'un développeur qui font qu'on aime mettre nos mains sur un clavier a évolué sur plusieurs générations, pas si sûr, tout du moins pour moi.
Pour répondre plus en détails à ces différentes questions, revenons un petit peu en arrière.
Si nos enfants sont des "digital native" comme le diraient les Anglo-Saxon, on peut dire que je suis un enfant de la Micro-Informatique. J'ai débuté la programmation dans les années 80 sur un IBM PC, je n'étais pas développeur mais Analyste-Programmeur d'ailleurs, ou avant d'écrire la 1ere ligne de code, je devais dessiner les étapes d'un algorithme sur une feuille de papier, à l'aide d'une règle spéciale du type qu'on utilise à la primaire aujourd'hui.

Je dois bien avouer que cela était assez scolaire, et me suis vite aperçu que personne ne faisait comme ça, et pour cause, quelle perte de temps! On a tous dans nos gènes la propension à démarrer tout de suite à vouloir écrire notre 1ere ligne de code. Alors oui, cet outil n'était pas forcément adapté à nos besoins et aux conditions dans lesquelles nous travaillions à l'époque. Toutefois, l'évolution des outils d'aujourd'hui qui en reprennent les mêmes principes, c'est-à-dire, d'avoir une meilleure vision globale de l'application et des interactions des différents éléments qui la composent, nous permettent de créer des applications plus robustes dans leurs fondements et plus robustes aux changements. Il ne faut pas hésiter à commencer à perdre un peu de temps au départ, pour en gagner beaucoup par la suite.

Une évolution, des outils, et des machines

Il est heureux que les outils aient évolué ou aient vu le jour, que de nouvelles méthodes de développement aient vu le jour, afin de nous accompagner à être plus productifs dans un monde où les enjeux, comme nous le verrons plus tard, ont évolué. Merci à tous ces développeurs et à tous ces architectes d'avoir mis leurs savoirs à disposition.
Comme nous étions sur des PC avec des ressources relativement restreintes, imaginez 640Ko de mémoire, voire 1Mo pour les plus chanceux, nous mettions tout en œuvre pour gagner le moindre octet et minimiser l'emprunte mémoire. La 1ere étape d'ailleurs, consistait à organiser le chargement des éléments du système d'exploitation dans un ordre bien défini pour perdre le moins de mémoire possible. Ensuite, en termes de développement, on grattait le moindre octet, en faisant attention à n'utiliser que le nombre exact d'octets dont nous avions besoin, et à bien libérer la mémoire.

C'est du bon sens de ne pas gaspiller les ressources, même aujourd'hui, mais qui s'en soucie avec des ordinateurs bourrés de Giga Octets de mémoires ? Qui se soucie aujourd'hui de l'ordre des données membres d'une classe ou d'une structure pour éviter le phénomène de padding pour l'alignement de la mémoire et gagner quelques octets ? Qui se soucie de la libération de la mémoire ? Est-ce que cela vaut d'ailleurs la peine de s'en soucier sur nos plateformes de développement et nos langages actuels qui le font très bien pour nous ?

Pour ma part, c'est devenu un réflexe Pavlovien, un réflexe conditionné qui m'oblige toujours à mesurer ce que je consomme. *C'est une réaction involontaire, provoquée par un stimulus extérieur* (en l'occurrence le manque de mémoire), *réaction non innée acquise par l'apprentissage et les habitudes qui deviennent des réflexes* Ivan Petrovitch Pavlov, mais cela ne peut pas être inné pour les jeunes développeurs d'aujourd'hui, même si dans une certaine mesure, nous le verrons plus tard, ils devront pour certains d'entre eux y passer.

A la même époque un nouveau périphérique révolutionnaire faisait son apparition, la souris, qui est devenue très vite un enjeu prioritaire. En effet nous avions à cœur d'être les 1ers à proposer à nos utilisateurs, une application qui gère ce dispositif de pointage. (Pour les plus curieux il fallait activer l'interruption 23h avec la valeur 3h. enfin je crois de mémoire).

Une des attentes fortes également des utilisateurs de l'époque, c'était de pouvoir imprimer leurs documents, et, un peu plus tard, en couleur s'il vous plait. Que ce soit sur des imprimantes matricielles (aiguilles), voire pour les plus fortunés, sur des imprimantes laser ou tous autres types d'imprimantes. Tout le challenge consistait donc, à proposer des interfaces de programmation unifiées sous forme de pilotes d'imprimantes, qui permettaient à nos applications de s'adapter aux différentes imprimantes.

Plus tard, si nous souhaitions des interactions réseau pour faire dialoguer deux applications entre elles, nous devions connaître et appréhender les 7 couches du modèle OSI (Open Systems Interconnection), afin que nos applications puissent, de manière transparente, s'envoyer des données sur le réseau.

Challenge après challenge

Avec l'avènement des interfaces graphiques, qui se soucie de la gestion des interruptions ? (à part les développeurs de drivers bien sûr), la souris s'est fondue dans le paysage et est devenue un élément tout à fait naturel. Qui se soucie d'adapter son application aux imprimantes ?
Qui se soucie des couches basses réseaux, alors que la majorité des échanges d'aujourd'hui se font sur le protocole http ?
Toutes ces contraintes, ces challenges, ont

plus ou moins disparu de la vision du développeur, et c'est tant mieux. Je suis content de l'avoir fait, mais tellement heureux de ne plus être obligé de le faire, et contrairement à ce qu'on pourrait vous dire, avant ce n'était pas mieux.

C'étaient nos challenges, nos enjeux de l'époque, et même si parfois je considère que nous avons une informatique de gosses de riches avec nos ordinateurs ultra-performants, gavés de mémoires, avec le recul, je m'aperçois de la futilité de mes propos.
Les enjeux d'aujourd'hui pour les développeurs se sont déplacés à un autre niveau d'abstraction; d'une certaine manière on doit prendre en considération d'autres notions essentielles, différentes, mais toutes aussi importantes dans le contexte actuel.

Alors évitons les anachronismes, ce que nous pensions indispensable il y a quelques années à connaître, n'est plus forcément d'actualité aujourd'hui ou s'est déplacé à un autre niveau. Seule la soif d'apprendre de nouvelles choses doit perdurer, afin de s'adapter à un monde en perpétuel mouvement.

Le développeur se doit d'évoluer, car tout s'accélère. La miniaturisation des composants a fait que les usages de l'informatique ont évolué, usages qui engendrent non seulement une explosion des échanges entre individus, que nos éminences grises identifient comme des océans de données à traiter et à sécuriser, mais qui engendrent aussi un niveau d'attente plus élevé de ces mêmes individus.

Si je prends comme exemple la catégorie des développeurs d'applications qui publient leurs applications dans un magasin quelconque, ou des applications dites mobiles, je ne suis plus sûr qu'ils aient encore du temps à consacrer à la manière dont fonctionne un processeur ou à la plomberie sous-jacente de telle ou telle technologie, de telle ou telle fonctionnalité. Ils ont à faire face à de nouveaux challenges, de nouvelles contraintes.

En effet, et ce qui est un peu antinomique avec ce que je vous ai dit plus haut, sur des ordinateurs ultra-performants, je dois l'avouer, c'est que certains appareils mobiles manquent de puissance, n'ont pas des ressources infinies, surtout pour les usages actuels. On ne parle plus en Kilo-Octets mais en Giga-Octets certes, mais il n'empêche que le développeur doit quand même faire attention aux ressources qu'il consomme. Que ce soit la consommation mémoire, mais aussi la consommation énergétique qui est un facteur différentiant. Qu'il doit adapter son application à une bande passante pas toujours de très bonne qualité. Comme tout cela n'est pas inné, le verdict de l'utilisateur est souvent cinglant.

Le bon code au bon moment

Pour gagner du temps, ils veulent donc trouver les bons exemples de codes, les bons services, les utiliser, les assembler avec d'autres qu'ils auraient piochés au gré de leurs recherches sur la toile et fournir le plus rapidement possible une application packagée; c'est d'autant plus exacerbé par l'explosion des appareils mobiles et leurs obsolescences programmées qui les poussent à le faire. Ils doivent aller vite au risque que leurs applications restent dans le peloton de queue dans un contexte éminemment concurrentiel.

A l'autre bout du spectre, le développeur, celui qui fournit des services, doit proposer des APIs, utiles et faciles à utiliser, sécurisées, résistantes aux attaques externes, qui évitent d'exposer inutilement les données utilisateurs en prenant en compte la confidentialité de leurs données, des Apis performantes (le temps d'attente de l'utilisateur s'étant amenuisé drastiquement avec le temps), et enfin qui évitent une surconsommation de ressources se traduisant, si elles tournent dans le Cloud, par une surfacturation.

Tous ces développeurs doivent faire face non seulement à ces nouveaux enjeux, mais également, pour certains d'entre eux, à des clients de plus en plus exigeants, qui ne veulent plus attendre la traditionnelle release de l'application. Le contexte évoluant, les clients évoluant, il faut bien que nos méthodes et nos outils évoluent pour y faire face.

Avec toutes ces évolutions, je dirais que dans une certaine mesure, ce métier est devenu plus complexe. Plus assisté par des outils informatiques et Internet certes, mais plus compliqué sans aucun doute, ou, plus précisément, qui demande de maitriser plus de concepts.

Ce que je veux dire par là, c'est qu'à mon époque j'ai appris l'informatique à un niveau d'abstraction assez bas, comme l'architecture de l'ordinateur et des processeurs au travers de livres comme celui d'Andrew Tanenbaum qui traite du réseau avec les 7 couches de l'OSI; j'étais confiné à un environnement complexe, bien évidemment, mais relativement simple et rassurant dans ses contours. Alors qu'aujourd'hui, le développeur doit appréhender des notions à un niveau d'abstraction beaucoup plus étendu,qui, pour certaines, dépassent les frontières de l'ordinateur traditionnel. Frontières ouvertes vers l'extérieur à tout le monde.

Prenons un exemple très simple

Qui dit frontière ouverte, dit problèmes de sécurité à tous les étages, même au niveau de la ligne de code. Lorsque j'ai débuté, ce n'était pas un réflexe (donc non inné), de faire attention à une attaque de type débordement de mémoire tampon dans son code. Mais avec l'ouverture des frontières, c'est devenu plus problématique et certains codes malveillants, ont su profiter de ces erreurs, même dans des applications de bureau traditionnelles. Alors oui, encore une fois, nous avons des systèmes et des plateformes de développement qui permettent de réduire la surface d'attaque. Mais le problème de sécurité reste entier et se déplace à d'autres niveaux. Un développeur qui expose son code sur la toile, doit faire face à de nombreuses attaques, de l'injection de script en passant par de l'injection SQL. J'en passe et des meilleures. Gardez bien à l'esprit que si le monde évolue, les attaques sur votre code évolueront aussi. Il est donc impératif, que le développeur évolue également dans ce sens. La sécurité n'est pas un sujet à prendre à la légère malheureusement et peu s'en préoccupent car elle induit toujours un surcoût significatif; vu de ma fenêtre, les coûts induits par le correctif, de l'écriture au déploiement, sont beaucoup plus importants.

Un autre exemple significatif des nouveaux challenges des développeurs, est l'explosion depuis 2005 du nombre de cœurs. Aujourd'hui au minimum nous avons 4 cœurs, même sur les smartphones c'est pour dire. C'est un véritable point d'inflexion que nous avons vécu à cette époque, car développer sur plusieurs processeurs, en parallèle, est une discipline complexe et réservée à un certain nombre d'experts très limité jusque-là. Aujourd'hui elle est essentielle pour chacun d'entre nous, car si une application n'exploite pas toute cette puissance théorique, elle a très peu de chance à mon avis, de percer dans un monde aussi concurrentiel.

N'étant pas une science innée, il faut que la majorité des développeurs s'approprient cette discipline. Fort heureusement et encore une

fois, nos plateformes de développement d'aujourd'hui fournissent des niveaux d'abstractions qui permettent de s'absoudre des arcanes des processeurs, et mêmes des arcanes de fonctionnement des systèmes d'exploitation; le véritable challenge pour le développeur qui développe majoritairement en séquentiel, c'est de se mettre dans un état d'esprit Penser parallèle James Reinders, avant d'écrire la moindre ligne de code.

"Développeurs nous avons tous la même passion, peu importe de l'instrument dont nous jouons".

L'étendue des concepts à appréhender est tellement gigantesque que les développeurs se spécialisent afin de devenir des experts dans telle ou telle discipline et échangent si besoin leurs compétences avec d'autres. Attention quand même à ne pas se laisser enfermer. Devenir un expert dans un domaine c'est bien, mais s'ouvrir à d'autres technologies, à d'autres plateformes, à d'autres langages et avoir plusieurs cordes à son arc et un plus indéniable.

La réalité fait qu'autour de moi, les développeurs se sont spécialisés pour fournir les meilleures applications et services possibles sur tous types d'appareils. Là est la beauté de ce métier, peu importe où vous vous trouvez dans le spectre des développeurs (spectre qui est beaucoup plus large que ce que je décris ici), c'est ce principe d'échanges et sa pluralité qui en fait toute sa grandeur. Les uns sont plus dans "l'ombre" et préfèrent développer des couches basses d'API utiles afin que d'autres développeurs, plus dans la lumière préfèrent créer des applications utiles et essentielles au plus grand nombre.
Alors qu'on soit dans la 1ere ou 2de catégorie ou dans n'importe quelle catégorie, que l'on utilise telles ou telles plateformes, tels ou tels langages, nous avons tous la même passion. C'est cette même passion qui nous anime depuis des générations lorsqu'on met nos mains sur le clavier, qu'on soit un développeur Cobol des années passées ou développeur Web d'aujourd'hui.

Un développeur reste un développeur dans sa mentalité, il doit avoir une éthique du travail bien fait, et doit être fier de son accomplissement. Alors bien évidemment j'ai rencontré différents niveaux de développeurs, mais quel que soit le niveau, du stagiaire au débutant en passant par le développeur confirmé, j'ai toujours vu dans leurs yeux cette étincelle qui me fait dire que nous avons le plus beau métier du monde. Bien sûr certains me rétorqueront, que pour une certaine catégorie de développeurs, c'est plus un tremplin, et que chef de projet est leur cible principale. Je n'en doute pas, mais au moins ils auront eu le mérite de mettre les mains dans le cambouis, eux !

"Le développeur est un créateur, qui, au bout des doigts, insuffle la vie à des entités inanimées."

C'est un peu, si je peux me permettre cette métaphore, comme Frankenstein qui assemble différentes parties de corps humains pour créer la vie. Mais attention à ce que sa création ne se retourne pas contre lui. Tous les développeurs en effet ne recherchent pas l'excellence loin sans faut, ceci sans doute pour différentes raisons qui se justifient, car viser l'excellence ou essayer de viser l'excellence n'est pas toujours forcement envisageable.

J'imagine que vous connaissez tous la règle des 80/20, et appliquée à la notion de performances elle prend tout son sens. Est-ce que je dois passer 80% de mon temps à optimiser une fonctionnalité alors que la majorité du temps se passe dans des échanges http que je ne maîtrise pas ? Si le ressenti général des performances de l'application est plutôt bonne, après avoir optimisé 80% du code, dois-je m'atteler à optimiser les 20% qui restent ? A vous de voir si vous en avez le temps. Il sera peut-être préférable de chasser les bugs pour faire en sorte que votre réalisation soit la plus parfaite possible, ou tout du moins d'éviter que les testeurs et les utilisateurs trouvent les bugs les plus basiques. C'est un minimum.

L'excellence c'est le graal absolu, et, pour ma part, et sans fatuité d'aucune sorte, c'est plus un état d'esprit dans lequel je me mets quotidiennement afin de livrer du code le plus abouti possible à un instant T.

J'ai un vieil adage, que tout travail doit être bien fait dès le départ, mais il m'arrive de revenir sur un vieux code de plusieurs années, voir même vieux de quelques semaines, mais quelle horreur m'exclame-je ! (et parfois je ne sais plus ce que j'ai voulu faire). Et pourtant j'en étais fier. Cette notion d'excellence est donc relativement abstraite, elle dépend pour beaucoup du contexte du moment, de l'expérience, des connaissances acquises et de ses propres références et critères. Mais si on a un regard critique sur notre passé, c'est que forcément, inexorablement, nous évoluons et c'est plutôt positif de penser que nous avons un métier qui nous pousse à nous dépasser.

De temps en temps dans ce métier il faut être pragmatique, il y a des applications dont le code, voire l'architecture laisse à désirer et ne reflète pas les canons de la beauté imposés par la mode du moment, et pourtant elles ont un succès fou !

Je sais que je vais en faire bondir certains, mais pour autant faut-il toujours succomber aux chants des sirènes ? Pas si sûr. Si par exemple, votre objectif c'est la performance à tout prix, et qu'en court-circuitant le Framework maison mis en place pour de nombreuses raisons justifiables, vous gagnez tout simplement de la performance, pourquoi ne pas être pragmatique dans ce cas-là ?

"Qui a décrété qu'à 50 ans si tu es encore développeur tu as raté ta vie ?"

Dans le monde professionnel, certains essaient de nous mettre une étiquette, pour essayer de nous comprendre. Mais je parlais plus haut de pluralité des métiers du développement, et il sera difficile de nous mettre dans des cases. Néanmoins ce qu'on voit depuis quelques temps, du fait que la technologie nous entoure et est présente quotidiennement, c'est que la vision du développeur auprès des autres évolue, jusqu'à nous coller l'étiquette de "héros" (j'abuse un peu mais on n'est pas loin). Ce qui est sûr, c'est que désormais certaines personnes ont pris conscience que derrière le monde numérique, qui devient chaque jour de plus en plus omniprésent et essentiel à leurs yeux, il y a des personnes de talent qui le construisent et les développeurs en font partie.

Le développeur prend une place de plus en plus prépondérante dans la vie de tous les jours de tout un chacun.
Du coup, on voit apparaître plusieurs initiatives Européennes autour de l'enseignement du développement et du code, que ce soient les écoles gratuites qui s'ouvrent à l'initiative des professionnels du secteur, en passant par des entrepreneurs qui montent une startup et qui ont besoin d'appréhender ce qu'est le développement, jusqu'aux plus petits où des professionnels prennent sur leur temps libre pour enseigner ce merveilleux métier dans les écoles. Je dois dire que cela fait du bien de vivre ça, et nous en avions sacrément besoin dans un pays ou on en manque cruellement et

où il a été délaissé depuis des années par manque de compréhension.

Alors, nous pourrions philosopher pendant des heures sur l'utilité de l'apprentissage du code à l'école, pour les plus petits ou les plus grands au collège ou au lycée, là n'est pas le débat et c'est un sujet qui a longuement été traité sur les réseaux sociaux. Mais de mon point de vue, je pense que c'est une discipline comme une autre à la différence près qu'il ne faille pas l'imposer afin de ne pas dégoûter de l'outil informatique. Mais bien enseigné par des personnes passionnées, il peut s'avérer un outil pédagogique formidable et susciter des vocations.

De mon expérience, même si dans mon Lycée nous avions les 1er Apple II, je n'ai pas débuté la programmation à l'école, et je le regrette. Que de temps perdu à cause d'un professeur d'informatique qui ne savait pas transmettre sa passion. Il a fallu un concours de circonstances et une chance inattendue pour que je puisse mettre la main sur un PC de l'époque. La programmation m'est venue tout naturellement et depuis ce temps-là je n'ai pas arrêté de m'y intéresser jusqu'à aujourd'hui.

On pourrait penser que depuis, une certaine routine s'est installée, que cela devienne ennuyeux et plus contraignant qu'autre chose (parfois c'est vrai). Mais si vous êtes passionné et curieux de nature, chaque jour est un nouveau jour ou vous avez de nouvelles choses à apprendre à appréhender, à dompter. C'est un métier avec un champ d'application tellement énorme, avec des possibilités infinies qui ont fait que je ne me suis jamais ennuyé. Apprendre stimule l'intellect. Bien évidemment, de temps en temps on se laisse dépasser par les évènements et on n'arrive plus à suivre, alors je me tourne vers mes pairs, des jeunes et des moins jeunes pour comprendre et partager avec eux. Nous avons la chance de faire un métier ou le partage n'est jamais à sens unique, ou on rencontre des développeurs exceptionnels qui vous poussent à vous dépasser et à vous remettre perpétuellement en question. Attention néanmoins aux personnes omniscientes ou qui voudraient vous le faire croire et il y en a, même chez les développeurs, si vous en rencontrez, vous n'avez qu'à les inviter un Mercredi à diner.

Si j'aime encore ce métier à 50 ans, c'est que depuis mes débuts comme je vous l'ai expliqué il a fortement évolué. Il a fortement évolué, parce-que le matériel a évolué, parce que les outils, les langages, les librairies, les services ont évolué. Il a fortement évolué parce-ce que le monde du numérique a évolué, monde omniprésent dans notre quotidien, qui est partout dans notre voiture, dans notre maison, sur nos vêtements, et même sur nous. On ne sait pas où et quand ce monde du numérique va s'arrêter, mais ce que je sais, c'est que j'ai la chance de faire un métier ou je peux transmettre mes acquis, et surtout, où ma courbe d'apprentissage est encore longue.

Qui peut se targuer de faire un métier qui propose des perspectives d'avenir infinies ? Je suis curieux de savoir ce que sera devenu le métier du développement dans 26 ans.

Les fondamentaux n'ont pas changé

Au risque de vous choquer, la base de la programmation informatique à mon humble avis n'a pas évolué de manière significative, au même titre que le moteur à explosion, si je peux me permettre ce parallèle. Moteur à explosion qui reste dans ses principes de bases depuis un siècle : on crée une étincelle d'une manière ou d'une autre, qui enflamme un carburant, qui engendre une explosion afin de faire monter et descendre un piston pour faire tourner un arbre à cames. En programmation c'est le même principe de base depuis des décennies. On code avec un langage de programmation, on compile le code en instructions machine, qui sont-elles mêmes exécutées par un processeur. Bien évidement il y a certaines variantes mais le principe reste le même.

Alors que sera le métier du développement dans 26 ans ? Est-ce que ses principes de base auront évolué ?

Est-ce que pour la plupart des développeurs on assemblera des boîtes, des services entre eux, sans écrire une ligne de code ?

Est-ce qu'on se mettra des électrodes sur le crâne pour capter les flux nerveux de notre cerveau afin de les retranscrire en instructions ?

Est-ce qu'on aura inventé l'ordinateur organique ? L'ordinateur quantique ?

Je n'en sais fichtrement rien, mais tout ce que je sais c'est qu'avant d'en arriver là, je vais encore avec bonheur mettre mes mains sur le clavier, car après tout, ne faisons-nous pas le plus beau métier du monde ?

mai

PHP Tour (Luxembourg) : 12 & 13 mai

Le PHP Tour s'arrête cette année au Luxembourg pour 2 jours. Le programme s'annonce copieux : programmation continue, coder son infrastructure, l'asynchronisme dans PHP, comment migrer dans le Cloud, , MySQL 5.7, gestion des erreurs, DDD, REST, déploiement avec Docker et AWS Beanstalk, Zend Framework 3...
Site : http://www.afup.org

Conférence NCrafts 2015

La conférence technique autour de .net et des langages Microsoft reviendra les 21 et 22 mai à Paris : plus de conférences, des ateliers dédiés, des speakers internationaux. Le plateau des intervenants sera très riche et de nombreux thèmes seront abordés (agenda non disponible à la parution de ce numéro) : agilité, debug, écrire un compilateur, TDD, continuous delivery, etc.
Site : http://ncrafts.io

juin

Matinale Club(21 : 4 juin

Rupture(21 vous invite à sa nouvelle matinale Club(21, curieuse conférence-atelier, pour découvrir et vivre des ingrédients d'une forme de leadership agile, essentiels pour réussir votre transition agile ou digitale. Pour plus d'informations.
event@zenika.com

Best of Web : 5 juin

Grâce aux nouvelles plateformes Web comme meetup.com, des communautés nouvelles ou existantes peuvent se retrouver et organiser des événements réguliers. C'est d'autant plus vrai chez les professionnels du Web qui ont créé de nombreux groupes parlant des derniers Frameworks Javascript, des API REST, de CSS et de Design. Ces meetups proposent toute l'année des conférences de qualité animées par des développeurs anonymes dans l'écosystème Web où se côtoient développeurs

chevronnés, consultants en freelance, étudiants et contributeurs open-source. Tellement d'événements ont lieu qu'il devient compliqué de pouvoir y assister, les rencontres se déroulant le soir en dehors des heures de travail. Huit groupes meetups ont donc décidé de se fédérer et de proposer un "Best Of Web" où le meilleur de leurs conférences sera rejoué sur une journée. Ces groupes portent sur des technologies variées du monde du web :

- AngularJS-Paris
- Backbone Paris
- Paris WebComponents
- EmberJs Paris
- Node.js Paris
- PhoneGap Paris
- D3.js Paris
- Paris JS

Les conférences qui ont été retenues comme "Best Of" sont déjà annoncées sur le site officiel de l'événement http://bestofweb.paris/ et le programme s'annonce très varié. Citons par exemple l'utilisation D'ES6 avec Angular par Douglas Duteil, une présentation de Polymer par Martin Gorner ou un panorama de l'écosystème REST par Virginie Bardales.
Mais ce n'est pas tout : de nouvelles présentations transverses seront choisies grâce au call for paper ouvert à la communauté.
Best Of Web se déroulera le 5 juin 2015 à l'Hôtel de Ville de Paris où 500 participants sont attendus.
L'ambition des organisateurs est de proposer l'évènement communautaire de l'année, participatif, sur un modèle différent de celui des conférences classiques où seules des stars du développement viennent s'exprimer. Cela sera à coup sûr le rassemblement d'une population de développeurs souvent expérimentés, toujours passionnés ayant en commun leur attachement aux technologies du Web.

Informations pratiques

site: http://bestofweb.paris/
Twitter: https://twitter.com/bestofweb2015
mail: bestofweb2015@gmail.com

WWDC 2015 : du 8 au 12 juin

La prochaine conférence développeur Apple sera très dense avec l'Apple Watch, des nouvelles de l'Apple TV, les nouvelles machines, les prochains iOS et OS X... Comme l'an dernier, Apple organise un tirage au sort. Plusieurs milliers de développeurs sont attendus, et des centaines d'étudiants. Comme d'habitude, la Pomme mobilise un millier d'ingénieurs...
Site https://developer.apple.com/wwdc/

Conférence UNITY, 24-25 juin

UNITE 2015 EUROPE

Unity, éditeur du moteur 3D, sera à Amsterdam les 24 et 25 juin pour sa grande conférence annuelle. Une occasion de parler techniques, de voir les dernières nouveautés et les futures évolutions mais aussi de discuter avec les partenaires. Il n'est pas trop tard pour prendre sa place !
Site : http://unity3d.com/unite/europe

Soirée JUG Paris

12 mai : Java / Azure
2 juin : design pattern vs lambda
23 juin : Tools in actions
site : http://www.parisjug.org/xwiki/bin/view/Main/WebHome

Interview de François Zaninotto, CEO de Marmelab

« Je m'appelle François Zaninotto, et je dirige Marmelab (www.marmelab.com), une startup d'une dizaine de personnes dédiée à l'innovation Web et Mobile pour les grandes sociétés. Je n'en suis plus à mon premier job : à 40 ans, j'ai déjà roulé ma bosse dans pas mal de grandes sociétés (Michelin, TF1) et des plus petites (la plus connue s'appelle SensioLabs). J'ai suivi un cursus ingénieur classique : classe préparatoire, puis trois ans à l'Ecole des Mines de Nancy de 1994 à 1997. A mon arrivée sur le marché du travail, j'ai débuté dans le domaine de l'Internet... et j'y suis toujours ! »

Comment es-tu tombé dans l'informatique et plus spécialement dans le développement ?

Vu mon âge, on va forcément parler d'antiquités ! Pour moi, ça a commencé un soir de 1984, lorsque mon oncle, qui logeait avec nous en banlieue parisienne, est rentré avec un paquet marqué « Canon X-07 ». Divorcé, pas très bon gestionnaire de son argent, il s'ennuyait beaucoup et avait donc acheté ce micro-ordinateur à écran à cristaux liquides intégrés (4 lignes, 20 colonnes), ainsi qu'une petite imprimante. L'ordinateur pouvait se programmer en BASIC, et, moyennant l'achat d'une carte d'extension de mémoire de 8Ko, exécuter des routines de dessin, et même des jeux. Mon oncle s'en est très vite lassé. Moi, j'ai très vite accroché !

J'ai commencé par recopier (à la main car Ctrl+C n'existait pas !) des listings trouvés dans des magazines, sans les comprendre. Ça prenait des heures. Pendant que je recopiais ces lignes de BASIC, j'imaginais l'expérience exceptionnelle que serait la mienne une fois le jeu lancé. Les illustrations en quadrichromie des magazines étaient forcément un peu enjolivées par rapport au rendu de l'écran monochrome de 120 pixels de large. Je suis malgré tout parvenu à faire tourner un jeu de plateforme, clone de Donkey Kong, en plus pixellisé. Après avoir rejoué les 4 mêmes niveaux une bonne centaine de fois, j'ai voulu en inventer de nouveaux, et j'ai retouché le code çà et là, à tâtons. Au passage, j'ai appris les variables, le binaire, les conditions, et la fonction magique : GOTO. A la fin, le jeu avait 30 niveaux et était digne d'une borne d'arcade (bon, j'exagère un peu !).

Et puis j'ai acquis un Amstrad CPC 6128, et enfin un Amiga 1000. Côté programmation, je suis passé au niveau supérieur, en développant un programme de visualisation de montagnes fractales en 3D. La 3D était naissante à l'époque; pour afficher un point avec des coordonnées dans l'espace sur un écran, il fallait maîtriser tous les calculs de projection dans l'espace et les modéliser pour l'ordinateur. Au début, n'y comprenant rien, j'ai demandé à mon professeur de maths, qui m'a trouvé des opérations matricielles tout droit sorties d'une thèse de fac. On n'aborderait les matrices qu'un an plus tard, alors j'ai dû élaborer mon propre algorithme, à base de géométrie plus sommaire (projections, vecteurs), mais que je comprenais. Une semaine après, le programme tournait. Une image fixe prenait une après-midi à s'afficher, mais j'étais très fier du résultat. Pendant une bonne année, je n'ai pas cessé d'améliorer mon programme, en y ajoutant de la couleur, de l'éclairage multi-points, de la neige, du brouillard, des formes arrondies, et une profondeur infinie grâce à la récursion.

Bref, côté développement, je suis autodidacte. Mais la programmation est une activité tellement prenante qu'il faut être vigilant à ne pas être tout le temps en train de penser à ses programmes. J'ai compris ça dès l'âge de 16 ans, et j'ai d'ailleurs arrêté complètement la programmation pendant 10 ans après le lycée. Ça m'a tout simplement permis de me construire.

J'y replongeais en 2005 avec Symfony, le framework PHP. J'ai collaboré à son élaboration, notamment en écrivant un guide d'utilisation (publié chez APress). Aux côtés d'un grand maître de la programmation (Fabien Potencier, l'auteur de Symfony), j'ai appris le développement objet, les design patterns, la programmation défensive, et les bonnes pratiques. Et j'ai surtout découvert les vertus de l'Open-Source : la collaboration ouverte permet de donner naissance à des assemblages fabuleux. Grâce à la programmation, je suis devenu Lead Developer sur des projets Open-Source (Propel, Faker, Uptime, et plus récemment Gremlins.js et ng-admin), formateur, conférencier, et je ne compte pas m'arrêter !

Aujourd'hui, je dirige Marmelab, une petite société dont le développement est le cœur de métier. Nous utilisons des outils de développement rapides Open-Source (comme Symfony2, React.js, Node.js) pour développer des prototypes innovants pour le compte de nos clients, en mode agile. Grâce au développement, j'ai pu créer 10 emplois, tous en CDI, et lancer des produits fascinants avec des grands noms du Web français. La demande est soutenue, et je continue à recruter des Architectes et des Développeurs Web expérimentés qui ont le goût du risque et la curiosité suffisante pour être à la pointe sur les nouveaux usages, les nouveaux business models et les nouvelles technologies.

Pour toi, qu'est-ce qui fait qu'on aime toujours et encore le développement, la technique ?

Je pense qu'on développe des programmes comme on joue aux Legos, parce que ça nous permet de donner naissance à des choses qui n'existent pas. Je considère la programmation comme une activité très créative. Mais également très prenante : une fois qu'on comprend que les possibilités sont infinies, l'imagination s'envole et ne s'arrête jamais.

Etre Développeur n'est pas toujours facile : pression, évolution constante, frustration des projets et des "chefs", c'est quoi pour toi d'être Développeur aujourd'hui ? Le job a-t-il évolué depuis tes débuts ?

Je suis l'évolution des techniques de développement Web depuis 15 ans, et la transformation du métier sur cette durée est ahurissante. Pour ne citer que les plus grosses révolutions, l'Open-Source, les frameworks MVC, la componentisa-

tion, l'asynchronisme et le déplacement de la logique côté client ont complètement modifié l'activité des développeurs.
En un sens, c'était bien plus simple il y a dix ans : on faisait tout dans l'entreprise, en utilisant un seul langage. Aujourd'hui, un Développeur doit savoir assembler des blocs existants plutôt que de réinventer la roue. Il doit maîtriser énormément de composants très divers (bases relationnelles, bases NoSQL, files d'attente, moteurs de recherche, frameworks full-stack, microframeworks, serveur Web, reverse proxy, frameworks front, outils de build, outils de déploiement, etc.). Il doit également se tenir à jour en permanence, parce que tout cela évolue très vite. Le métier de Développeur est donc plus exigeant aujourd'hui, mais il permet de faire des services et des logiciels bien plus ambitieux.
A mon sens, la transformation récente la plus importante du métier de Développeur tient non pas à la technique, mais aux méthodes agiles. Elles ont réussi à sortir les Développeurs du modèle client-fournisseur pour les faire devenir acteurs du changement. Les Développeurs sont les partenaires du Product Owner, et cela change tout. Mais attention : « With great power, comes great responsibility ! » Quant à la pression, elle augmente dans tous les métiers. Je trouve que les Développeurs sont plutôt favorisés par rapport au reste de la population.

En dehors du boulot, qu'est-ce que tu aimes faire ? Comment trouves-tu l'équilibre entre travail, vie privée, passion, famille ?

Il y a sept ans, j'ai trouvé un super truc pour occuper mes loisirs : ça s'appelle les enfants ! Pas besoin de se poser la question en rentrant du travail : la famille accapare 100% de mon attention. Et je ne suis pas pressé de leur apprendre à programmer : je préfère les activités face à face que tournées vers un écran. Les quelques heures que j'arrive à dégager dans la semaine, je les passe avec mes amis, ou bien à faire du sport (jogging, randonnée, aïkido) et du piano.

Peux-tu nous présenter ton quotidien en quelques mots ?

Je suis Chef d'entreprise, mon quotidien est donc très peu répétitif. Une fois passées les activités de stratégie, de suivi projet, de coaching technique, de démarchage commercial, de recrutement, de veille, de management et de gestion, il me reste quelques heures par semaine pour faire du développement et de la R&D. D'ailleurs, ces deux activités se confondent pour moi : mon temps de développement est exclusivement attribué à des projets Open-Source, prospectifs, qui me permettent de progresser sur un domaine particulier ou d'accélérer les développements de l'entreprise par des outils réutilisables.
Dans mon précédent poste, je passais 80% de mon temps en réunion ! J'ai réussi à réduire la proportion à moins de 50%, mais mon quotidien reste très marqué par le travail en équipe. C'est un besoin du métier, et un goût personnel.

Comment vois-tu ton job évoluer ?

Après quinze ans, j'ai acquis quelques certitudes sur les choses que je ferai toujours, et celles que je ne ferai plus jamais. Je sais par exemple que je ferai toujours du développement, et pour deux raisons : parce que j'aime ça, et parce que ma crédibilité auprès de mon équipe et de mes clients en dépend. Je passe plus de temps aujourd'hui à faire de la revue de code et du pair programming que du développement seul ; je sais que mon job continuera à évoluer ainsi, vers davantage de travail en équipe et de direction de projet. Mon activité technique s'oriente vers l'architecture logicielle. L'accumulation des expériences permet de prendre du recul et de mieux prévoir l'évolution des systèmes. C'est pourquoi les Architectes sont souvent des Développeurs qui ont beaucoup de bouteille.
Il y a des choses que je ne ferai plus parce que je peux les déléguer (suivi comptable et juridique notamment), et des choses que je ne ferai plus parce que je n'aime pas ça (par exemple gérer une équipe de 50 personnes, faire un projet en cycle en V). Mais ce qui est important, c'est qu'en tant que Chef d'entreprise, j'ai la liberté de choisir où je veux mener ma carrière. Cette liberté a une valeur énorme à mes yeux. Elle me permet de faire évoluer mon job en fonction du plaisir d'abord, et de la rentabilité ensuite.

Des conseils aux étudiants et Développeurs qui nous lisent ?

C'est un âge d'or pour les Développeurs en ce moment. Tout le monde a besoin de vous, vous pourrez choisir les meilleures sociétés et négocier rapidement de très bons salaires. Mais j'ai vécu la crise de 2001, et notre société est elle-même en crise grave. Ce n'est pas parce que vous êtes indispensables aujourd'hui que vous le serez demain. Mon conseil numéro un, c'est donc : ne prenez pas la grosse tête. J'ai vu trop de jeunes diplômés se comporter comme des divas et montrer des exigences farfelues. Le métier de Développeur reste un métier de technicien, et un Développeur seul ne peut rien faire. Sachez vous intégrer dans une équipe, et trouver votre place en gagnant le respect de vos pairs et de vos clients.
Mon second conseil, c'est de ne pas s'arc-bouter sur une technologie en particulier. Vous pouvez être fans de Ruby on Rails, React.js ou Django, mais rien ne vous permet de dire que les autres technologies sont moins bien. Et vous ne voulez pas devenir les Développeurs COBOL de 2017. Le dogmatisme est, en informatique comme en religion, une plaie. Un langage n'est pas bon ou mauvais en soi, il est adapté ou pas à un contexte. Les langages que j'utilise aujourd'hui en production n'existaient pas il y a trois ans. Cela veut dire que les langages que j'utiliserai dans trois ans n'existent probablement pas encore. Il n'y a qu'en étant pragmatique, humble et curieux qu'on peut durer dans ce métier.
Et un dernier conseil, c'est que l'évolution de carrière que beaucoup de Développeurs souhaitent à court ou moyen terme, vers la gestion de projet, est souvent une impasse. D'abord, un Chef de projet est souvent moins bien payé qu'un Développeur expérimenté. Ensuite, les méthodes agiles permettent aux Développeurs de s'impliquer dans le produit et suppriment la nécessité d'un intermédiaire avec le client. Enfin, pour effectivement gérer des projets, il faut des qualités et des envies bien différentes de celles qui mènent au métier de Développeur. Il y a plein de suites de carrières exaltantes pour un Développeur : Architecte, Lead Developer, Formateur, Expert technique, Directeur technique. Chef de projet est à mon sens la moins bonne des options. ⊞

MON BUREAU

Mon environnement de travail est très mobile, et tient donc dans un MacBook Air et un smartphone. Je suis là où sont mes collaborateurs, ou bien là où sont mes clients. Mais l'environnement que je réserve à mes collaborateurs est, lui, plus typique d'une startup du Web. Dans les locaux de Marmelab Paris, que nous partageons avec PALO IT (www.palo-it.com), cabinet de conseil en innovation technologique, il y a des MacBook Air, plein de salles de réunion, des Barcamps mensuels et du mouvement en permanence. Dans les locaux de Marmelab Nancy, on trouve de grands tableaux blancs, plein d'écrans, un baby-foot, un canapé, un décodeur Canal+, un frigo rempli, et de la bonne humeur. Malheureusement, mes collaborateurs s'obstinent à se lancer des défis musicaux atroces (comme écouter « Dominique Nique Nique » en boucle pendant trois jours !) pour tester leur capacité de concentration dans des conditions extrêmes, et nos visiteurs nous regardent de travers. C'est dommage, car à part ça, ils sont très agréables au quotidien !

Dmitry Gritskevich, développeur Java chez Oodrive

Le 12 mars dernier, l'école 42 organisait un grand concours de développement, rassemblant pas moins de 700 développeurs, qui se sont affrontés tout au long de la soirée pour déterminer quel était le meilleur d'entre eux. Sur la deuxième marche du podium s'est hissé un jeune Russe, Dmitry Gritskevich, développeur au sein de la société Oodrive.

Dmitry Gritskevich a grandi en Russie, à Kogalym, une petite ville perdue au milieu de la Sibérie occidentale, à plus de 3 000 kilomètres à l'est de Moscou. Kogalym n'est pas vraiment l'endroit où l'on rêve de passer son enfance : l'hiver dure plus de 6 mois, et la température moyenne annuelle avoisine les - 5°C.

« La vie là-bas était très difficile, explique Dmitry. Quand on est un petit garçon, on ne peut pas aller jouer dehors avec ses copains, par exemple. Il faut donc trouver des occupations chez soi. C'est ainsi que j'ai découvert l'informatique. Mes parents m'ont acheté mon tout premier ordinateur, je crois que c'était un Vecteur C. »

L'informatique apparaît rapidement à Dmitry comme une vocation, et un bon moyen de s'envoler vers des climats plus cléments. Après plusieurs années d'études à Moscou, il décroche son premier emploi d'ingénieur qualité, et découvre les interfaces Java. Il a seulement 24 ans quand il quitte la Russie pour rejoindre Paris, en 2010...

« C'est en France que je suis vraiment devenu développeur... Je ne parlais pas français et Oodrive m'a pourtant fait confiance. Mon manager s'adressait à moi en anglais ; il m'a permis de m'intégrer au sein de l'entreprise et de continuer d'apprendre mon métier. »

Au sein d'une équipe d'intégration, qui intervient directement chez le client afin d'installer les solutions Oodrive de partage de fichiers, Dmitry cultive ses facultés d'adaptation et apprécie la polyvalence de sa tâche : installation des serveurs, configuration des réseaux, développement d'espaces Cloud privés... Il cumule aujourd'hui les fonctions de développeur Java et d'ingénieur sécurité licence (pour les clients en externe).

« Chaque client comporte ses propres spécificités, s'enthousiasme Dmitry. C'est donc vraiment enrichissant, d'être toujours confronté à de nouvelles exigences et à de nouveaux défis. Il s'agit de comprendre la demande des clients et de proposer des choses qui répondent exactement à leurs besoins. Cela implique de ne pas rester bloqué sur sa propre vision des choses, et de faire preuve d'ouverture d'esprit. »

L'adaptation est sans doute la clé de la réussite de Dmitry. En arrivant à Paris depuis sa Sibérie natale, il a été confronté à une nouvelle langue, un nouveau climat, un nouveau travail, de nouvelles méthodes et de nouveaux matériels. Loin de constituer pour lui des obstacles, ces épreuves ont enseigné à Dmitry en toutes circonstances. Une qualité qui s'est avérée décisive lors du concours du Meilleur Développeur de France organisé par l'École 42, où Dmitry s'est distingué en se plaçant en seconde position.

« Le vainqueur Antoine Leblanc trouve que le concours n'a pas tout à fait récompensé le meilleur développeur de France, mais le plus rapide. Je suis assez d'accord avec lui. L'École 42 nous a imposé de travailler sur des ordinateurs Mac, dotés d'écrans de très grande taille, et avec des claviers QWERTY, ce qui a pas mal déstabilisé la plupart de mes adversaires. Le challenge était donc de s'adapter le plus vite possible... »

42 - DR –© Laurent Belando

LE MEILLEUR DEV DE FRANCE

Si Dmitry reste humble après cette deuxième place méritante – *« J'ai fait exprès d'arriver 2e : je me suis dit que les Français auraient un peu honte de voir un Russe devenir le meilleur développeur de France »*, plaisante-t-il – la société Oodrive peut s'enorgueillir de compter parmi ses équipes de développement les meilleurs éléments de la profession. Le leader du Cloud européen compte plus de 90 développeurs parmi ses effectifs, et organise chaque année des « hackathons » au sein de ses locaux. Un événement qui fait l'unanimité :

« Le hackathon, c'est notre fête. C'est une façon de nous exprimer, un événement qui nous est propre, à nous développeurs. C'est une excellente initiative qui motive tout le monde, qui ressoude les équipes... Et le vainqueur de notre hackathon a ensuite l'opportunité de participer au concours du Meilleur Développeur de France. C'est donc à Stanislas de Rémur (CEO d'Oodrive) que nous devons tout cela, et je tiens à le remercier... »

« L'avantage de travailler chez Oodrive, ce sont les nombreuses opportunités d'évolution et d'enrichissement personnel que l'entreprise cultive au quotidien. Beaucoup de mes collègues ont débuté leur carrière en tant que développeurs HTML, et travaillaient sur des fonctionnalités de base, avant de devenir gestionnaires de projets ou responsables clients. Oodrive nous permet d'identifier nos points forts et de les faire fructifier... Je connais même des gens qui ont commencé au sein du service marketing et qui travaillent maintenant au service IT. Parce que c'est ce qu'ils préfèrent... »

Bienvenue dans l'économie des API

11-03-13 © Kirillm / iStock

On parle depuis environ 2 ans d'économie des API, dans le sens où les API dirigent une partie des entreprises et du business. Cette approche dépasse largement les développeurs. L'API est désormais un outil, comme un autre, pour les entreprises, les banques, les administrations, etc. Mais fondamentalement, l'API ne change pas de nature. Elle change de dimensions... Plusieurs éléments sont mis en avant pour expliquer cette tendance : aider l'innovation, accroître la productivité, développer de nouveaux de canaux et marchés, mieux interagir et échanger avec son environnement (entreprises, prestataires, fournisseurs, etc.). Pour une entreprise, l'API permet d'exposer des services, des fonctions et pour une autre entreprise, de les consommer et de les utiliser plus rapidement et plus simplement. L'API fournit, théoriquement, tous les éléments nécessaires. Par exemple, une entreprise propose d'exposer des données concernant des horaires de spectacles dans toute la France (en open data). La création d'une API Web va permettre à un développeur d'une app mobile de se connecter à ces données sans forcément connaître le modèle de données derrière, ni l'infrastructure qu'il y a. Il récupère l'API, l'appelle et code la glue pour se connecter et récupérer la ou les données nécessaires.

Le succès, et l'intérêt, d'une API dépend de plusieurs éléments : la qualité de celle-ci, la valeur qu'elle peut générer, la simplicité, la maintenance (une API doit vivre), le support technique, sa licence.

Dans ce dossier, nous allons démystifier les API !

La rédaction

Développer facilement vos APIs Web avec APISpark

Une API Web est un ensemble de ressources accessibles sur le Web via des URIs, et exposant des opérations sous forme de méthodes HTTP standards, ainsi qu'un ensemble de représentations définissant la structure des messages échangés. Elle permet notamment de travailler sur les données indépendamment de leur format tels que JSON, XML ou YAML grâce à la négociation de contenu HTTP. Les APIs Web sont intimement liées aux concepts de REST, le style d'architecture du Web.

Thierry Templier (ttemplier@restlet.com)
Lead Architecte, Restlet SAS
Co-auteur des ouvrages Restlet in Action (Manning) et Spring par la pratique (Eyrolles).

APISpark est une plateforme Cloud (PaaS) permettant de créer, d'héberger et de gérer rapidement et simplement des APIs Web en mettant à disposition un environnement de développement intégré en ligne (IDE Web) ainsi qu'un service d'hébergement. Elle permet de définir des APIs Web en se basant sur les concepts de REST. Son IHM guide le développeur afin d'exposer des ressources et des traitements derrière une URI. La puissance de l'outil réside dans le fait qu'aucune ligne de code n'est nécessaire puisque les interactions entre les APIs Web et leur backend peuvent être configurées visuellement.

Concepts des APIs Web

Les APIs Web se basent sur les concepts de REST (REpresentational State Transfer), un style d'architecture ciblant les systèmes distribués dans le contexte du Web. Un des concepts clé est l'absence d'état au niveau du serveur. En effet, chaque requête d'un client vers un serveur doit contenir toute l'information nécessaire pour permettre au serveur de comprendre la requête. Bien que cette architecture ne dépende pas directement du protocole HTTP, ce dernier est particulièrement adapté pour la mise en œuvre de ces concepts. Avant de détailler la manière de définir graphiquement son API Web dans APISpark, nous allons rappeler les principaux concepts de

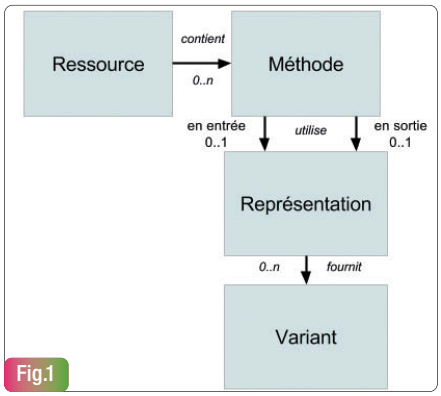


Fig.1

REST. Le concept central de REST est la ressource. Cette dernière peut être vue comme un élément qui gère un type particulier de données. Elle fournit ainsi différents traitements pour gérer son état. Il est par contre possible de définir plusieurs ressources pour gérer un même type de données : par exemple, une pour la liste des éléments et une autre pour l'élément unitaire. Une ressource est accessible sur Internet à partir d'une URL et il est possible d'interagir avec elle avec un ensemble de méthodes prédéfinies afin de récupérer l'état associé, de le mettre à jour et d'exécuter éventuellement d'autres traitements. Avec HTTP, nous pouvons tirer parti des différentes méthodes (aussi appelées verbes) que le protocole fournit. Le format des données échangées correspond à la représentation. Il s'agit d'une structure logique de données n'étant liée à aucun format particulier tel que JSON, XML ou YAML. Cette structure peut posséder des attributs avec des niveaux d'imbrication. Le contenu d'une représentation avec un format particulier est appelé variant. La Fig.1 illustre les concepts décrits ci-dessus ainsi que leurs relations.

HTTP fournit un ensemble de méthodes avec une sémantique précise qu'il convient de respecter pour être conforme à l'architecture REST. Prenons l'exemple des deux ressources permettant de gérer un contact : une au niveau de la liste de contacts (avec l'URL /contacts/) et l'autre au niveau d'un contact particulier (avec l'URL /contacts/{contactId}). Nous décrivons ci-dessous l'utilisation classique des méthodes HTTP dans ce contexte :

- **Méthode GET.** Elle permet de récupérer l'état d'une ressource tel qu'une liste de contacts si utilisée sur la ressource de type liste ou un contact en particulier sur la ressource unitaire.
- **Méthode POST.** Elle permet d'ajouter un contact si utilisée sur la ressource de type liste.
- **Méthode PUT.** Elle permet de remplacer complètement les données d'un contact si utilisée sur une ressource unitaire.
- **Méthode PATCH.** Elle permet de faire une mise à jour partielle de l'état de la ressource. Elle est typiquement utilisée pour des mises à jour en masse sur la ressource de type list et pour modifier uniquement certains champs sur la ressource unitaire.
- **Méthode DELETE.** Elle permet de supprimer un contact particulier.

La Fig.2 décrit ces différentes méthodes des ressources utilisables pour gérer des données. Maintenant que nous avons vu les principes des APIs Web, nous pouvons aborder la manière dont la plateforme APISpark favorise leur mise en œuvre d'une manière simple.

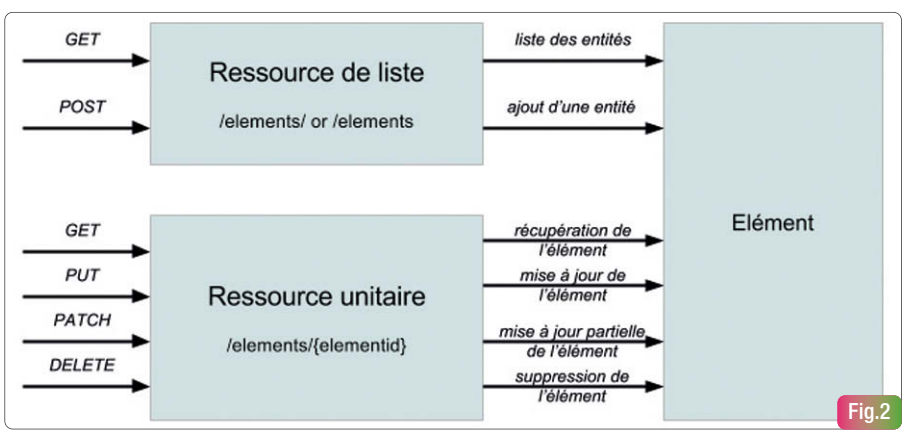


Fig.2

Définir son API Web

APISpark (http://restlet.com/products/apispark/) met à disposition différents assistants de création (accessibles en cliquant sur la baguette magique en haut à droite) dans différents contextes :

- Création d'APIs Web visant à gérer des données,
- Création d'entités de supervision d'APIs Web,
- Création d'API Web de documentation.

Nous allons nous focaliser ici sur le premier cas et allons créer puis héberger une API Web de gestion de données structurées. Nous verrons

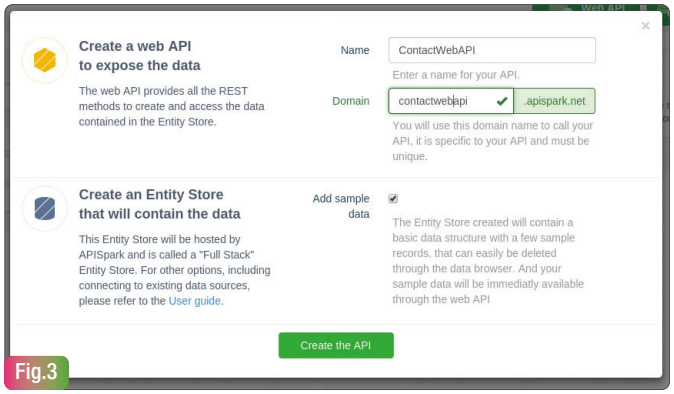


Fig.3

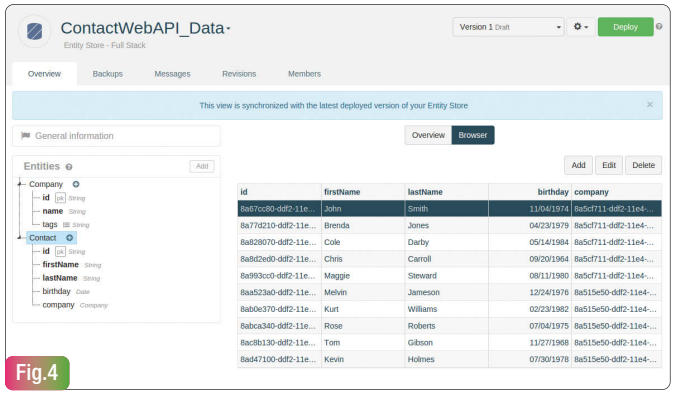


Fig.4

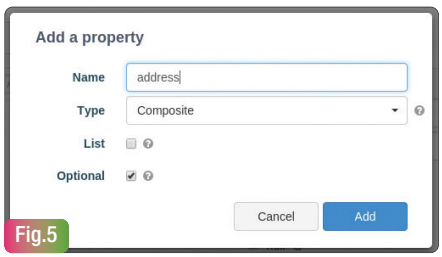


Fig.5

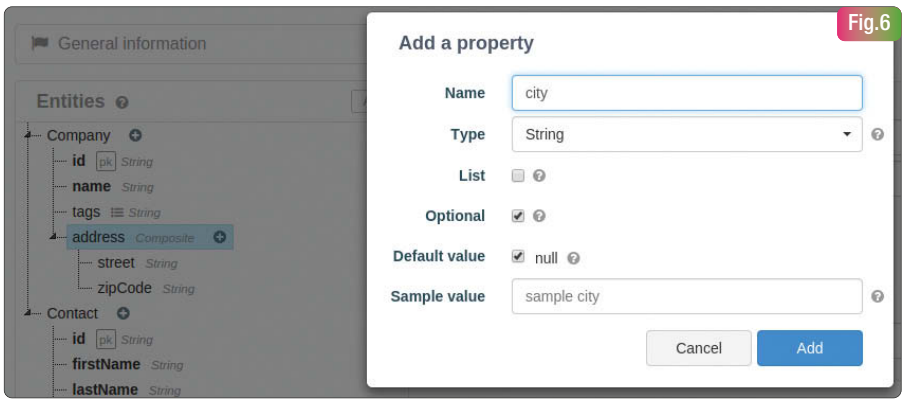


Fig.6

par la suite les APIs de documentation. L'assistant de création d'APIs Web de gestion de données nous permet très simplement et rapidement de créer une API fonctionnelle. Il suffit de saisir le nom de son API Web ainsi que son domaine pour y accéder, comme l'illustre la Fig.3 suivante. L'assistant crée alors deux éléments appelés cellules, une de type "Entity Store" avec le nom "ContactWebAPI_ Data" pour la gestion des données et une de type "Web API" avec le nom ""ContactWebAPI" pour l'API Web en elle-même qui exposera ses données via HTTP. Comme nous avons coché la case "Add sample data" (ajout de données d'exemples), des structures de données sont créées automatiquement en base avec un jeu de données ainsi que le montre la Fig.4. Deux entités, "Company" et "Contact" ont été créés avec différentes propriétés dont notamment une relation entre les deux via une propriété "company". Il est facilement possible de modifier cette structure de données en utilisant le menu de gauche de l'application ainsi que la zone centrale d'édition. Nous pouvons ainsi rajouter, modifier ou supprimer des champs. APISpark supporte à ce niveau trois types de champs distincts: les types primitifs, les types composites et les références. Nous pouvons par exemple ajouter à l'entité "Company" une propriété composite "address" contenant son adresse (rue, code postal, ville), ainsi que le montrent les Fig.5 et 6 suivantes. Nous avons désormais fini la mise à jour de notre structure. Il nous faut maintenant aller sur notre cellule de type "Web API" pour la synchroniser avec cette nouvelle structure. En effet, l'assistant a lié nos deux cellules lors de la création. Ainsi notre cellule l'API Web dépend de celle de type "Entity Store". Cet aspect est visible dans ses

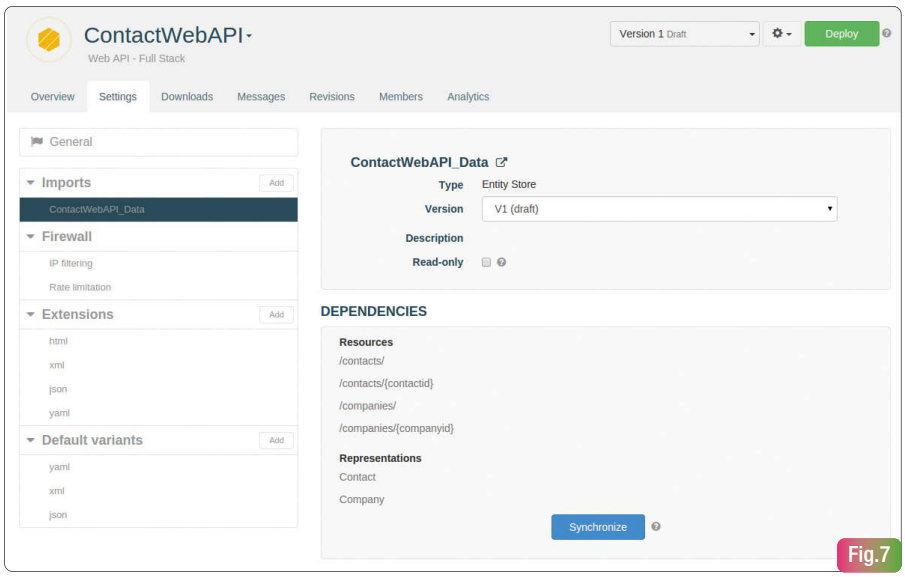


Fig.7

paramètres (onglet "Settings", section "Imports") qu'illustre la Fig.7. Il nous suffit de synchroniser la dépendance vers l'Entity Store pour que l'API Web soit calée avec le modèle mis à jour. Déployons nos deux cellules afin que les changements soient utilisables. Nous voyons bien dans le menu de gauche les différentes représentations calquées sur nos précédentes entités ainsi que les ressources associées et les différent es opérations utilisables. En peu de temps, nous avons ainsi réalisé une API Web opérationnelle permettant de gérer nos données. Il est à noter que ce type d'API Web ne peut être modifiée que partiellement puisque sa structure est directement déduite de celle des données. Il est uniquement possible de supprimer des ressources et des méthodes que l'on ne souhaiterait pas rendre accessibles. Nous ne décrirons pas plus les possibilités offertes par APISpark mais il faut savoir que la plateforme intègre d'autres fonctionnalités avancées comme une gestion des droits d'accès, du cache serveur pour améliorer les performances, ainsi que des restrictions d'appels (nombre de requêtes maximum pour une période donnée), etc.

Utiliser l'API Web

L'API Web est alors accessible directement depuis le domaine spécifié lors de sa création. Les informations complètes telles que son URL et les informations de sécurité sont accessibles

depuis la zone "Endpoints" dans le menu de gauche, ainsi que le montre la Fig.8. Une fois ceux-ci identifiés, nous pouvons aller au niveau des représentations, afin de voir les structures de données disponibles, et des ressources pour les méthodes utilisables et les structures échangées aussi bien en entrée qu'en retour. Par exemple, pour notre représentation "Company", la Fig.9 illustre un exemple de format des données avec le variant JSON. Nous somme désormais prêts pour faire notre premier appel à notre API Web. Commençons par une requête d'ajout d'une société ("Company"). Nous pouvons utiliser n'importe quel client REST. Par simplicité, nous allons exécuter nos requêtes en nous fondant sur l'extension Postman dans Chrome. Exécutons tout d'abord une méthode POST sur l'URL https://contactwebapi.apispark.net/v1/companies/ avec la structure d'exemple récupérée précédemment afin d'ajouter une société, ainsi que le montre la Fig.10. APISpark génère un identifiant unique et renvoie le contenu complet des données ajoutées. Cette société a été ajoutée dans la base de données sur laquelle se base l'API Web. Nous pouvons ensuite utiliser une méthode GET sur la même URL afin de constater qu'elle est désormais présente dans la liste des sociétés comme l'illustre la Fig.11. De la même manière, nous pouvons utiliser toutes les méthodes des ressources définies au niveau de l'API Web. Elles permettent notamment de gérer un élément en particulier afin de le récupérer, de le mettre à jour et éventuellement de le supprimer. Il est à noter qu'APISpark fournit une intégration très fine avec Swagger UI.

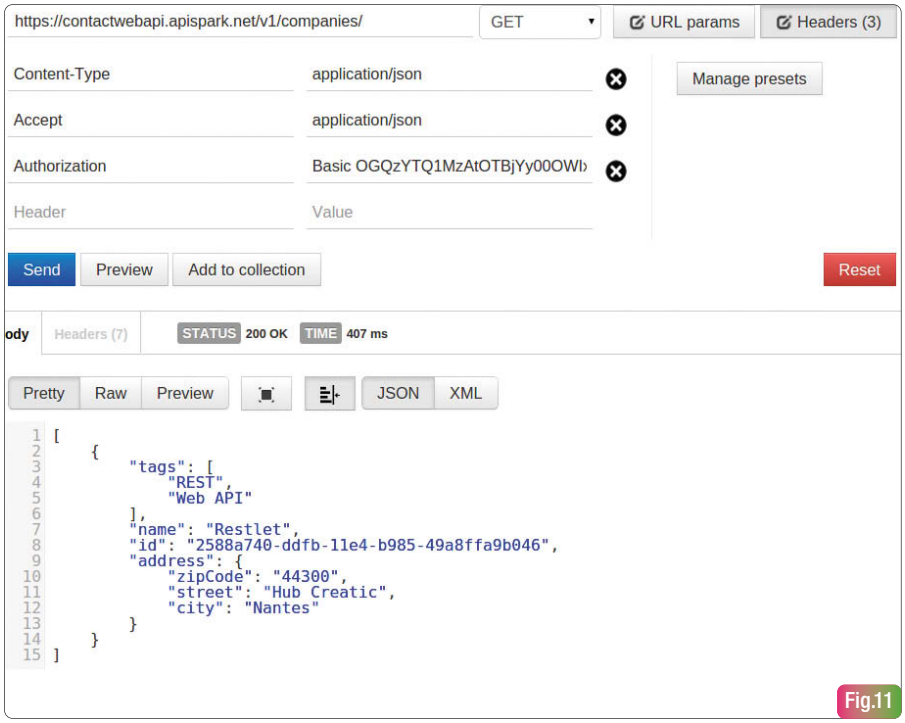


Fig.11

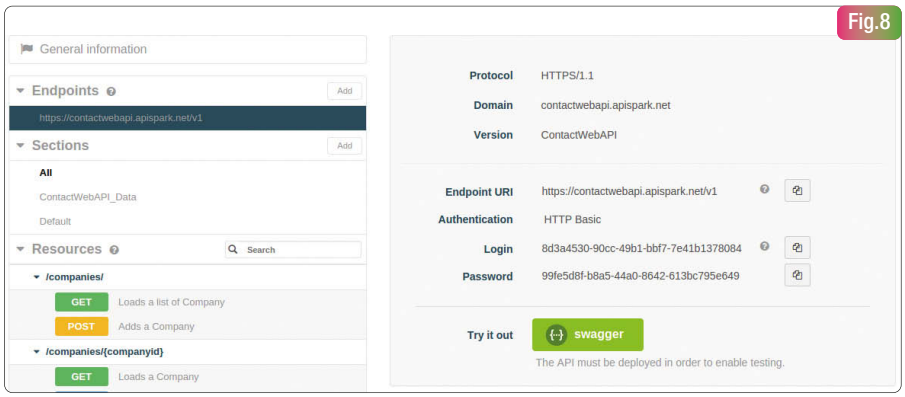


Fig.8

Fig.9

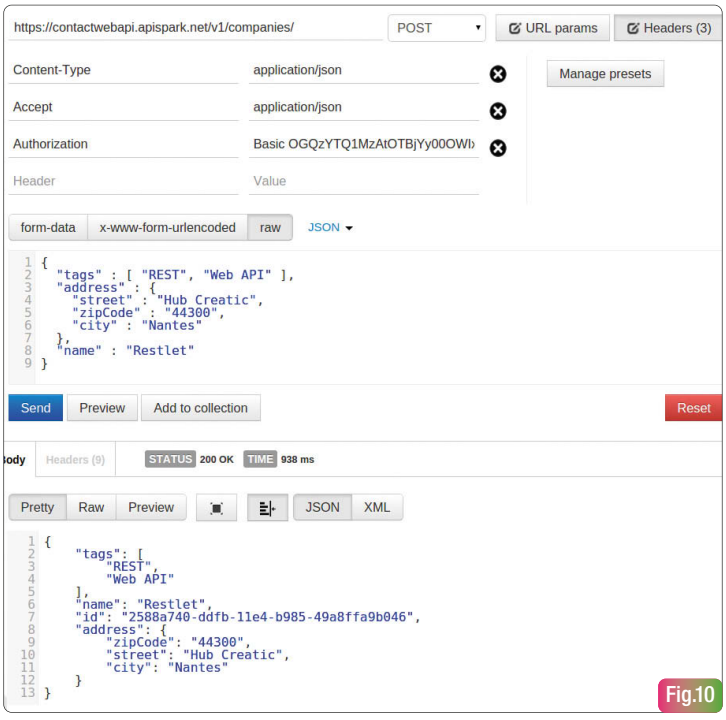


Fig.10

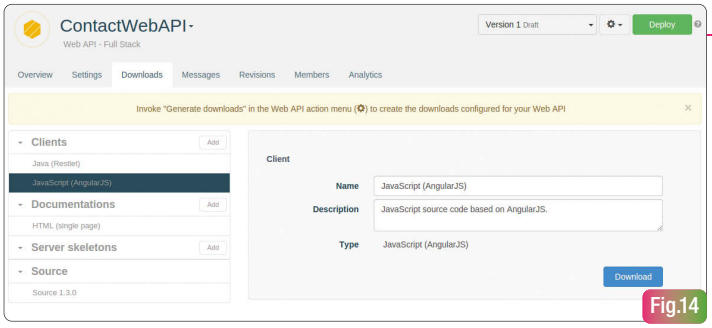


Fig.14

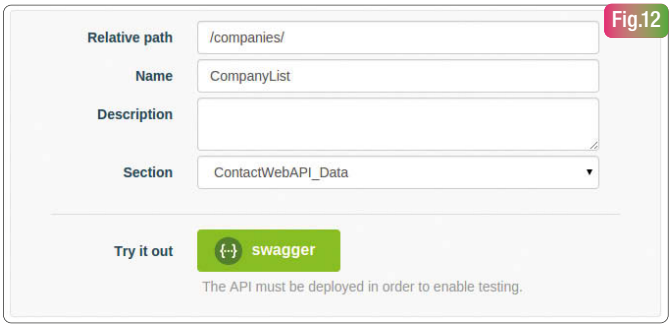


Fig.12

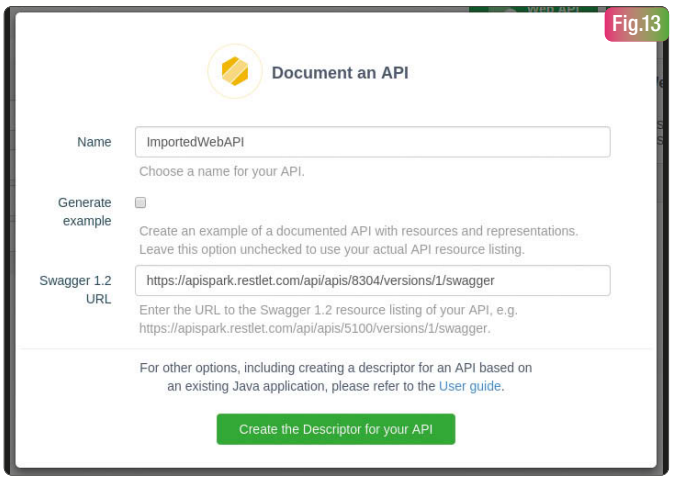


Fig.13

Cette dernière permet de visualiser les APIs Web et de simuler des appels sur différentes méthodes de ressources et tout cela en ligne. Les bons formats sont préremplis en se basant sur les données saisies dans APISpark et les paramètres d'URLs supportés sont affichés afin de leur positionner une valeur. A cet effet, un bouton « Try it out » est disponible à différents endroits de l'interface Web pour afficher l'application et afficher le bon contexte. La Fig.12 montre ce bouton au sein des propriétés d'une ressource. Dans ce cas, Swagger UI s'affichera, et la ressource correspondante sera sélectionnée.

Importer des définitions de Web API existantes

Il existe dans le monde des API Web plusieurs formats décrivant leurs structures. Nous pouvons distinguer notamment:

- Swagger (http://swagger.io/) qui propose une définition au format JSON,
- RAML (http://raml.org/) qui utilise quant à lui un format YAML.

Bien qu'APISpark permette de générer ces différents formats à partir d'APIs Web vprécédemment configurées dans l'outil, il est également possible d'utiliser ces formats pour créer son API Web dans la plateforme. Les avantages de cette approche consistent en la possibilité de créer en ligne, et simplement, aussi bien la documentation correspondante, que ses clients programmatiques et ses squelettes serveur pour différentes technologies (Java, JavaScript, Android, ...). Nous allons prendre ici le format Swagger 1.2 correspondant à notre précédente API Web pour illustrer notre propos mais n'importe format Swagger 1.2 disponible sur le Web peut être utilisé. Pour utiliser cette fonctionnalité, il suffit d'exploiter l'assistant de création d'API Web de type documentation. Ce dernier prend en paramètre le nom de l'API ainsi qu'une URL permettant d'accéder au contenu Swagger dans le but d'initialiser la structure de l'API. La Fig.13 décrit cet assistant. Dans ce cas-là, il est possible par la suite de faire évoluer librement la structure de l'API contrairement à celle orientée données. Une autre différence réside dans le fait qu'une telle API n'est pas destinée à être déployée dans la plateforme APISpark. Elle ne peut donc pas être exécutée ni appelée.

Clients programmatiques

Une API Web adresse les communications inter-applications et il est donc commun de l'appeler depuis une application et avec un certain langage. Redéfinir les traitements d'appel constitue réellement une perte de temps puisqu'ils peuvent être aisément déduits. D'une manière similaire, l'implémentation d'une API Web peut être rendue plus simple et rapide à partir d'un squelette de code. La plateforme APISpark fournit en standard cette fonctionnalité en se basant sur la structure des APIs Web définies. A cet effet, différents types de contenu peuvent être générés puis téléchargés :

- Clients programmatiques tels que Restlet pour Java ou Android / Restlet, iOS et Angular dans le contexte de JavaScript,
- Squelettes d'application tels que JAX-RS pour Java, NodeJS pour JavaScript et Scalatra pour Scala,
- Documentations (HTML),
- Source Restlet Framework de l'API Web.

A cet effet, un onglet « Downloads » est mis à disposition. Une fois tous les contenus générés, ils peuvent être accédés depuis le menu de gauche et téléchargés séparément à partir des panneaux de détail, ainsi que l'illustre la Fig.14.

Conclusion

APISpark est une plateforme Cloud (PaaS) mettant à disposition un IDE Web productif permettant de créer et documenter facilement et rapidement des API Web ainsi qu'un service d'hébergement intégré afin de les mettre à disposition sur Internet. Dans cet article, nous avons montré comment développer une API Web de gestion de contacts avec APISpark, exposant des données structurées. Nous avons vu comment créer une API Web à partir d'un modèle de données en laissant l'outil créer les ressources et représentations correspondantes. Nous avons également vu que la source de données est facilement synchronisable avec les éléments exposés par l'API Web, et que leur déploiement se réalise en un seul clic avec un bouton Deploy. Vous n'avez plus à vous soucier des problématiques d'hébergement, de disponibilité et de scalabilité : la plateforme les gère pour vous. Une autre fonctionnalité intéressante d'APISpark consiste dans son intégration avec l'outil Swagger UI afin d'interagir graphiquement avec l'API Web déployée dans la plateforme. Cela permet facilement de tester son API et de voir concrètement les données échangées. Nous avons également décrit comment importer et documenter une API Web dont la structure est disponible au format Swagger. Cela permet d'avoir accès à des contenus générés tels que des clients programmatiques, des squelettes d'application serveur ainsi que de la documentation. Nous n'avons pu aborder dans cet article que les fonctionnalités majeures de la plateforme. D'autres fonctionnalités telles que la possibilité de superviser des APIs Web sont également disponibles. L'ergonomie de l'interface Web et des aides contextuelles facilitent son utilisation. Des tutoriels ainsi qu'une documentation utilisateur étoffée sont également mis à disposition respectivement aux adresses http://restlet.com/technical-resources/apispark/tutorials et http://restlet.com/technical-resources/apispark/guide.

Langages de description des APIs Web

Thierry Templier

Définir le contrat d'une API Web est sans doute un des aspects les plus importants lors de sa mise en œuvre. En effet, ce contrat régit notamment les ressources mises en œuvre, leurs méthodes d'accès ainsi que les structures et formats échangés. Cette phase est loin d'être une perte de temps car, en se basant sur des langages de description, elle permet de gagner du temps par la suite dans les développements et dans la création des documentations associées. Pour autant, faut-il choisir le bon outil à ce niveau afin de pouvoir tirer parti de ce travail en amont de l'implémentation? Il y a eu ces dernières années une explosion du nombre de langages permettant de spécifier et de décrire la structure d'APIs Web. Il n'y a pas vraiment actuellement de standard qui émerge. De plus, les technologies des APIs évoluant rapidement, le choix du langage correspondant au mieux aux besoins reste une tâche ardue. Néanmoins trois langages de description semblent plus populaires au sein de la communauté des APIs Web. Ils fournissent tous la possibilité de décrire simplement les différents constituants d'une API Web ainsi qu'un outillage adapté afin de les visualiser, les éditer, les parser et de générer du code dans différents langages pour consommer ou implémenter ces APIs Web.

- Swagger (http://swagger.io/) basé sur le format JSON est un standard de fait de par sa popularité et l'activité de sa communauté.
- RAML (http://raml.org/) est un langage basé sur le format YAML. Des outils annexes (dont un éditeur) sont développés par la société MuleSoft.
- API Blueprint (https://apiblueprint.org/) est un langage sponsorisé par la société Apiary et basé sur le format Markdown. Cela le rend très facile à lire et à définir pour un utilisateur "humain" (à opposer à une application).

Nous pouvons également citer un dernier langage qui se distingue des trois précédents car il est principalement orienté documentation.

- Slate (https://github.com/tripit/slate) permet de générer de très belles documentations d'APIs incluant des exemples d'appels client dans plusieurs langages, associés à la structure de l'API.

Supportant plusieurs langages et apportant donc une unification entre ceux-ci, Restlet Studio (https://studio.restlet.com/) est un autre outil intéressant à ce niveau. En effet, il permet de définir graphiquement et en ligne la structure de son API Web ou de l'importer à partir d'un contenu Swagger. Restlet Studio offre ensuite la possibilité de télécharger différents formats tels que Swagger et RAML et de générer en ligne des kits (SDK) clients pour interroger l'API correspondant au contrat ainsi qu'une base de code (un squelette) pour développer une API implémentant cette structure.

QUESTIONS – RÉPONSES AVEC GREG BRAIL (APIGEE)

Greg Brail est architecte en chef d'Apigee, éditeur d'une plateforme complète dédiée aux API.

Les API explosent aujourd'hui, pourtant l'API n'est pas une nouveauté. Comment l'expliquer ?

S'il est vrai que les API ont fait partie de l'écosystème d'Internet depuis un certain temps, cela ne fait que quelques années qu'elles sont très présentes. Les API sont très efficaces pour être utilisées dans les apps mobiles afin de communiquer avec un service backend. Elles ont grandi très rapidement. D'autre part, le modèle de l'API est si facile à utiliser et à comprendre que nous voyons des entreprises les utiliser dans leurs propres systèmes informatiques. La plupart des développeurs comprennent l'API.

Il existe plusieurs langages de descriptions des API comme RAML. Pourquoi le développeur a-t-il besoin de maîtriser ces langages pour développer une API ?

RAML, Swagger, et les autres, sont des langages ayant un usage spécifique qui peuvent être utilisés pour décrire une API d'une manière « lisible » pour les machines. En décrivant une API de cette manière, il est possible de générer automatiquement une documentation (technique) de l'API, et il existe des outils pour créer rapidement une API ou un client pour une API. Une API est un ensemble de services basés sur http; de nombreux outils « parlent » http et peuvent être utilisés pour créer l'API, et il n'y a dans ce cas pas besoin d'apprendre un nouveau langage.

La sécurité est toujours un sujet sensible. Qu'en est-il de la sécurité et des API ? Qu'est-ce que le développeur doit faire ou mettre en place ?

Comme tout service basé sur des services réseaux, la sécurité prend différents aspects, l'API n'y échappe pas. Il y a l'authentification (quand on appelle l'API), l'autorisation (par ceux qui sont autorisés à le faire) et la détection des menaces (le client tente d'envoyer une forte charge non valide). Bien entendu, il y a d'autres considérations telles que les politiques de gestion de trafic pour prévenir qu'un client a fait plus d'appels qu'autorisés. Aujourd'hui, OAuth 2.0 est un standard utilisé pour l'authentification et l'autorisation. Il est très flexible et peut s'intégrer avec de nombreuses autres technologies.

Apigee n'est pas l'unique outil pour créer et gérer les API. Quelles fonctions vous différencient ?

Les différenciants sont de répondre aux besoins de l'entreprise et des missions critiques, d'être capable de supporter des milliards d'appels API par semaine et d'être focalisé sur le service, la facilité d'utilisation par le développeur.

Quelles bonnes pratiques le développeur doit avoir pour créer et maintenir les API ?

Vaste question ! Ces dernières années, plusieurs bonnes pratiques se sont créées. Par exemple, toujours vérifier à la fois l'usage final d'une API et l'application qui y fait appel. Toujours avoir un moyen de contrôler comment les nombreux appels API sont faits par les applications, vérifier les demandes entrantes non conformes ou non valides. Ne pas oublier de suivre tout ce qui vient de l'API afin de parfaitement comprendre comment l'API est (réellement) utilisée.

Coder sa première API avec python 1ère partie

Dans cet article nous proposons de développer votre premier service Web exposant une API orientée RESTful à l'aide de python. Nous décrirons la pile logicielle minimum nécessaire pour lancer votre solution en production. Nous verrons comment configurer un serveur Linux afin qu'il accueille ce service en configurant un reverse proxy (nginx) ainsi qu'un contrôleur de processus (supervisord). Première partie : les bases.

Aurélien Moreau
Co-fondateur d'Angus.ai – Human Perception for Machines.
Blog: http://www.angus.ai/blog/

Le but de ce tutoriel est de présenter les différentes couches à mettre en place pour fournir un service web et ainsi offrir une API pour des applications clientes. Nous allons mettre en œuvre un service qui propose une solution au problème du sac à dos (*knapsack* en anglais). A travers l'API, le client proposera une liste "d'objets" et la capacité limite d'un sac à dos. Le service répondra un sous-ensemble de la liste qui maximise un gain. Il est intéressant de noter que le monde des API web ou services web (nous utiliserons ces terme de manière indifférenciée par la suite) est un mélange de programmation (le métier du service) et de scripting/configuration/administration système (le métier de l'exploitation). Nous proposons dans cet article d'effectuer toutes ces tâches pour des raisons pédagogiques, mais des plateformes existent et vous offre la possibilité d'exécuter directement votre code métier en mode Plateforme As A Service. De manière plus technique nous allons développer l'algorithme en python, le transformer en service web en utilisant *Tornado* un framework web python. Ce service sera ensuite par un superviseur bien connu dans le domaine python *supervisord*. Pour conclure nous protégerons cette solution derrière le reverse-proxy *nginx*. Tout au long du développement nous nous efforcerons de garder en tête la philosophie RESTful, de manière à proposer un service qui sépare bien les concepts de client et de serveur. Nous ferons un service sans état, et les résultats seront potentiellement mis en cache. De plus nous cacherons au client la présence des différentes couches (*nginx*, *Tornado*) et finalement nous communiquerons avec les verbes classiques HTTP et une représentation uniforme JSON indépendante de la représentation python Fig.1.

ENVIRONNEMENT

Première chose à faire est d'installer son environnement de travail. Nous utiliserons virtualenv et pip :

```
> virtualenv env
> source env/bin/activate
> mkdir knapsack-api
> cd knapsack-api
```

COMMENCONS PAR LA DOC

Pour ce tutoriel nous avons choisi le problème du sac à dos qui consiste à trouver la meilleure combinaison d'objets, au sens de l'utilité, à prendre dans un sac à dos sachant que chaque objet a un poids, une valeur d'utilité et un nom le tout avec un poids limite à ne pas dépasser. La figure Fig.2 décrit les structures de données manipulées. Une API est avant tout une documentation qui tient lieu de contrat entre le client et le fournisseur de service. Cette documentation peut être rédigée de différentes manières. Dans le cadre de ce tutoriel nous utiliserons tout simplement sphinx.

```
> pip install sphinx tornado
> mkdir doc
> cd doc
> sphinx-quickstart
```

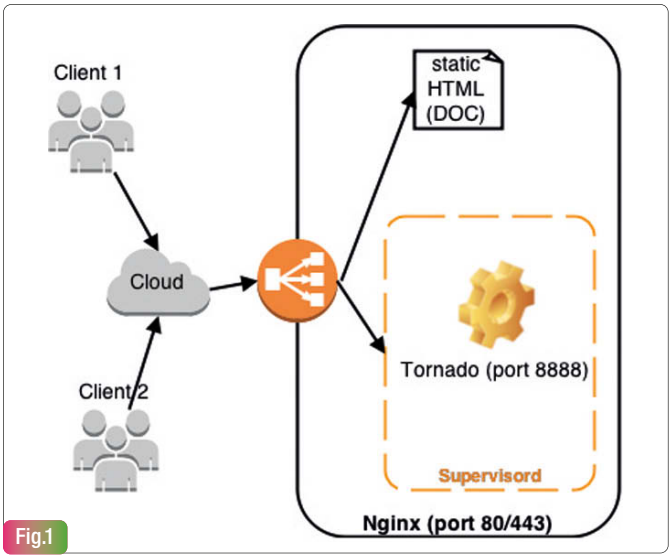


Fig.1

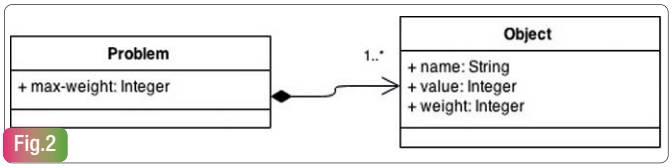


Fig.2

Il suffit alors d'éditer le fichier index.rst puis d'y déposer un début de documentation.

```
Welcome to knapsack-api's documentation!
========================================
Post a problem
--------------
**POST** /api/problems

Description
+++++++++++
Submit a new problem to the resolver

Request body
++++++++++++
Example::

 {
  "objects": [
    { "name": "o1", "value": 5, "weight": 2},
    { "name": "o2", "value": 8, "weight": 3},
    { "name": "o3", "value": 14, "weight": 5},
    { "name": "o4", "value": 6, "weight": 2},
    { "name": "o5", "value": 13, "weight": 4},
    { "name": "o6", "value": 17, "weight": 6},
    { "name": "o7", "value": 10, "weight": 3},
    { "name": "o8", "value": 4, "weight": 1}
```

```
  ],
  "sack": {
    "max-weight": 12
  }
 }

Response body
+++++++++++++
Example::

 {
  "link": "http://mydomain.org/api/problem/0",
  "objects": [
    { "name": "o8", "value": 4, "weight": 1},
    { "name": "o7", "value": 10, "weight": 3},
    { "name": "o5", "value": 13, "weight": 4},
    { "name": "o4", "value": 6, "weight": 2},
    { "name": "o1", "value": 5, "weight": 2}
  ],
  "sack": {
    "weight": 12
  }
 }
```

Une fois terminée on peut générer une documentation statique de notre API et l'exposer sur le web. Ceci constitue la première étape essentielle.

```
> make html
```

La figure Fig.3 présente le résultat. Comme la documentation le présente bien, nous sommes dans le cas d'un échange client / serveur traditionnel. La création d'une nouvelle ressource *problem* est indépendante de tout contexte d'exécution donc *stateless*. Il n'y a aucune information relative à un intermédiaire quelconque et finalement nous proposons une représentation sous forme de JSON avec le vocabulaire classique HTTP. Pour toutes ces raisons le service peut être considéré comme RESTful.

UN CLIENT DE TEST

Avant même de développer le service, nous sommes maintenant capables de créer un client, toujours en python sur la base de la documentation. Ce dernier va nous permettre de tester par la suite notre API. Nous allons utiliser python requests et écrire un script très simple, créant un problème et récupérant le résultat.

```
> pip install requests
```

```
if __name__ == "__main__":
 data = {
  "objects": [
   { "name": "o1", "value": 5, "weight": 2},
   { "name": "o2", "value": 8, "weight": 3},
   { "name": "o3", "value": 14, "weight": 5},
   { "name": "o4", "value": 6, "weight": 2},
   { "name": "o5", "value": 13, "weight": 4},
   { "name": "o6", "value": 17, "weight": 6},
   { "name": "o7", "value": 10, "weight": 3},
   { "name": "o8", "value": 4, "weight": 1},
  ],
  "sack": {"max-weight": 12}
 }
```

Le principe est simple, un premier appel créé une nouvelle ressource *problem* avec une méthode POST dont le corps correspond à la description des paramètres comme la documentation le définit. On passe d'une structure de données python à du json et vice et versa grâce aux fonctions *json.dumps()* et *json.loads()*.

```
url = "http://localhost:8888/api/problems"
result = requests.post(url, data=json.dumps(data))
result = json.loads(result.text)
print(json.dumps(result, indent=4))
```

Le lien vers la ressource est récupéré lors de la création et est utilisé dans un second appel, cette fois-ci un simple GET pour vérifier que tout c'est bien passé.

```
link = result["link"]
result = requests.get(link)
result = json.loads(result.text)
print(json.dumps(result, indent=4))
```

LE SERVICE

Une fois ce travail préliminaire effectué nous pouvons nous attaquer véritablement au développement du service. Pour ceci nous allons installer tornado :

```
> pip install tornado
```

Une façon de traiter le problème du sac à dos est d'utiliser un algorithme de programmation dynamique. Le cœur de la solution est codé dans la fonction resolve() qui prend en argument les données du problème.

```
def resolve(items, sack):
 max_w = sack["max-weight"]

 v = [[(0, False)] * (max_w+1) for i in xrange(len(items))]

 for w in xrange(max_w+1):
  if w < items[0]["weight"]:
   v[0][w] = (0, False)
  else:
   v[0][w] = (items[0]["value"], True)

 for k in xrange(1, len(items)):
  for w in xrange(max_w+1):
   if w < items[k]["weight"]:
    v[k][w] = (v[k-1][w][0], False)
   else:
    old = (v[k-1][w][0], False)
    new = (v[k-1][w-items[k]["weight"]][0]+items[k]["value"], True)
    v[k][w] = max(old, new)

 result = []
 w = max_w
```

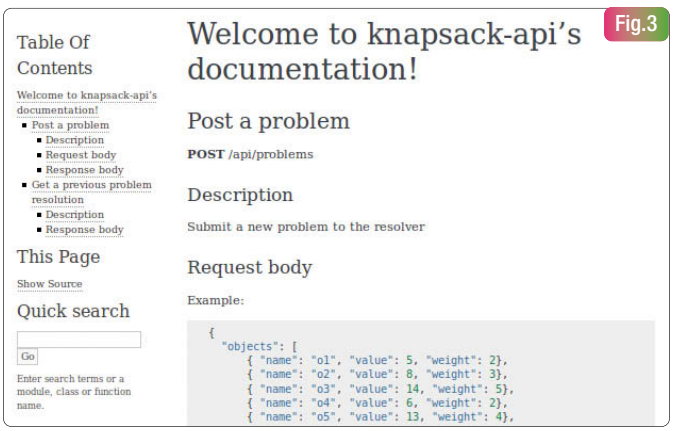


Fig.3

```
for k in xrange(len(items)-1, -1, -1):
  if v[k][w][1]:
    result.append(items[k])
    w -= items[k]["weight"]

return result
```

Deux "handler" sont définis, un pour l'ensemble des problèmes, il est enregistré pour répondre aux requêtes sur l'url /api/problems. L'autre est enregistré pour répondre aux requêtes sur l'url /api/problems/[id] ou id est un entier correspondant à un numéro de problème attribué par notre service lors de la création. Tornado nous permet alors de surcharger les fonctions post et get respectivement pour les deux handler. Par ailleurs nous utilisons la surcharge d'un initialiseur propre à Tornado qui permet de référencer un espace de stockage (pour garder les résultats des problèmes). Pour faire simple nous utilisons ici une solution « en mémoire » à travers un dictionnaire. Le handler de création de problème peut donc alors faire appel à la fonction resolve et stocker le résultat dans le storage passé à l'initialisateur Tornado.

```
class ProblemsHandler(tornado.web.RequestHandler):
  def initialize(self, storage):
    self.storage = storage

  def post(self):
    data = json.loads(self.request.body)
    prob_id = len(storage)
    result = dict()
    storage[prob_id] = result
    url = "http://localhost:8888/api/problems/%s"%(prob_id)
    result["link"] = url
    result["objects"] = resolve(data["objects"], data["sack"])
    result["sack"] = data["sack"]
    self.write(json.dumps(result))
```

Comme nous avons stocké les résultats dans un dictionnaire dont la clé est l'id du problème, rien de plus simple pour les retrouver, la méthode get est triviale.

```
class ProblemHandler(tornado.web.RequestHandler):
  def initialize(self, storage):
    self.storage = storage

  def get(self, prob_id):
    self.write(json.dumps(self.storage[int(prob_id)]))
```

Une fois les *handlers* achevés et enregistrés, il suffit de lancer la boucle d'exécution de Tornado en écoutant sur un port, ici 8888 :

```
storage = dict()
application = tornado.web.Application([
    (r"/api/problems", ProblemsHandler, dict(storage=storage)),
    (r"/api/problems/(.*)", ProblemHandler,  dict(storage=storage)),
])

if __name__ == "__main__":
    application.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

Après avoir lancé le serveur, on peut immédiatement utiliser le client en local (en utilisant *http://localhost:8888*) pour tester notre nouveau service :

```
> python application.py &
> python client.py
```

MISE EN PRODUCTION

Pour l'instant nous avons un service qui se lance à la main, et qui écoute sur le port 8888. Il nous faut donc encore le passer en "production". Pour cela nous allons utiliser le superviseur *supervisord* qui nous permet de lancer, monitorer et daemoniser notre service, ainsi que *nginx* qui va nous permettre de l'exposer au monde sur le port 80 en toute sécurité.

SUPERVISOR

```
> pip install supervisor
```

Une fois installé, il suffit de décrire son application pour que *supervisord* s'en charge :

```
[unix_http_server]
file=/tmp/supervisor.sock

[supervisord]
logfile=/tmp/supervisord.log
loglevel=info
pidfile=/tmp/supervisord.pid
nodaemon=false

[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock

[program:knapsack-api]
command=$VIRTUALENV/bin/python application.py
process_name=knapsack
numprocs=1
```

Il suffit alors d'éxécuter supervisord pour que notre service se lance. Nous pouvons alors vérifier que tout s'est bien passé en exécutant de nouveau le client.

```
> supervisord -c supervisord.conf
> python client.py
```

NIGINX

La configuration nginx n'est pas plus compliquée.

```
server {
  listen 80;
  location ~ ^/api(.*)$ {
    proxy_pass http://127.0.0.1:8888/api$1$is_args$args;
  }
}
```

Cette configuration minimale écoute sur le port 80. Elle capture toutes les requêtes commençant par */api* pour les rediriger sur le serveur interne grâce à l'instruction *proxy_pass*. La configuration capture la suite de l'url *(.*)* et la recopie vers la destination *$1*, elle en profite également pour copier d'éventuels paramètres *is_argsargs*.

CONCLUSION

Nous avons déposé les premières briques fonctionnelles d'un service web exposant une API sur le net afin qu'il puisse être utilisé par tout à chacun. De nombreuse amélioration peuvent, et doivent être apportées à ce premier Hello World. Nous verrons ceci dans un prochain article.

Swagger : un langage de description d'API

Cyrien Autexier
co-créateur de l'application mobile Foodbie et full-stack développeur à Geek Learning Studio. Geek Learning Studio est un groupe d'experts passionnés de développement et de technologie.
@geek_learning - http://geeklearning.io

Le monde des APIs et des architectures orientées services a longtemps été marqué par l'omniprésence de SOAP. Ce standard reposant sur http et XML, apportait déjà la promesse de services auto descriptifs pour lesquels il était facile de générer une documentation et des clients. Toutefois, il s'avérait verbeux, sa norme était complexe, et les implémentations introduisaient souvent des spécificités nuisant à l'interopérabilité... Progressivement, les développeurs et particulièrement les startups, se sont détournés de ce standard bien trop lourd et complexe pour leurs besoins. Ce sont les services RESTful qui ont pris le dessus. REST constitue un retour à la base du protocole http et à ses fonctionnalités (content negociation, compression, ssl/tls). L'explosion des usages mobiles a amplifié ce mouvement à un tel point que REST est aujourd'hui omniprésent.
Toutefois, il existe pratiquement autant de façon de faire du REST que d'API. Par exemple, certaines sont basées sur les erreurs http, alors que d'autres vont toujours renvoyer des codes 200, et utiliseront une enveloppe dans le corps du message pour indiquer le statut de la requête. Au final, REST est une jungle où chacun fait un peu ce qu'il veut. Là ou auparavant il suffisait d'utiliser un générateur et le wsdl pour obtenir un client de service, il faut aujourd'hui parfois plusieurs jours pour comprendre la documentation d'un service et l'utiliser. D'une certaine manière c'est une véritable régression.
Naturellement, des initiatives pour normaliser et documenter les services REST sont apparues. L'une des premières est probablement OData. OData apporte les métadonnées décrivant les schémas de données et les opérations possibles. Toutefois, OData s'avère relativement complexe et est très impactant sur la forme finale de l'API.

Les différentes initiatives

De ces constats, plusieurs initiatives ont vu le jour. Les plus prometteuses aujourd'hui sont probablement Raml, Blueprint et Swagger. Swagger propose un ensemble d'outils et de bibliothèques permettant de couvrir tous les aspects depuis la conception jusqu'à la génération d'api en passant par la documentation et le live test.

- Swagger Specification : L'objectif de « Swagger specification » est de définir un standard pour les APIs REST qui permette aussi bien aux développeurs qu'aux machines de comprendre et utiliser ces services. La spécification se veut agnostique par rapport au langage ou à la stack utilisée. « Swagger spec » définit donc un schéma pour des métadonnées (YAML ou JSON) pour la description des apis REST. Une des forces de l'approche de Swagger par rapport à ODATA par exemple, c'est sa grande souplesse et sa prise en main rapide.
- Swagger Editor : permet d'écrire ou visualiser des spécifications de manière simple et efficace grâce au live preview « human readable ». De plus il intègre les autres outils de Swagger permettant la génération de squelettes serveur et de librairies clientes.
- Swagger UI : Swagger-UI est une bibliothèque HTML/JS permettant de générer une documentation et un playground à la volée à partir de métadonnées Swagger.
- Swagger generator : c'est un générateur permettant de produire du code serveur ou client répondant aux spécifications Swagger. De très nombreux langages sont d'ores et déjà supportés.

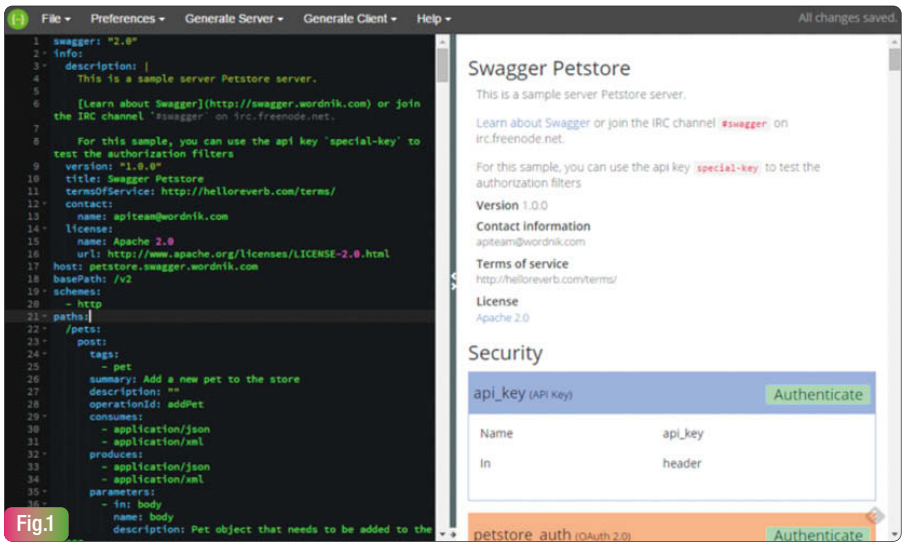


Fig.1

Description d'une Api

Pour découvrir plus en détail comment ça se passe, le plus efficace c'est tout simplement de lancer Swagger Editor et d'ouvrir l'exemple « Swagger Petstore » Fig.1.
Le fichier d'exemple est le suivant :

```
swagger: "2.0"
info:
  description: |
    This is a demo API
  version: "1.0.0"
  title: Demo API
  contact:
    name: apiteam@demo.example.com
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
host: demo.example.com
basePath: /v2
schemes:
  - http
  - https
paths:
  /contact:
    post:
      tags:
        - contact
      summary: Add a new contact
      description: ""
      operationId: addContact
      consumes:
        - application/json
      produces:
        - application/json
      parameters:
        - in: body
          name: body
          description: Contact to add
          required: false
          schema:
            $ref: "#/definitions/Contact"
      responses:
        "200":
          description: Contact added
        "405":
          description: Invalid input
  security:
```

```
    - api_key: []
  put:
    tags:
      - contact
    summary: Update an existing contact
    description: ""
    operationId: updateContact
    consumes:
      - application/json
    produces:
      - application/json
    parameters:
      - in: body
        name: body
        description: New contact information
        required: false
        schema:
          $ref: "#/definitions/Contact"
    responses:
      "200":
        description: Contact updated
      "405":
        description: Validation exception
      "404":
        description: Pet not found
      "400":
        description: Invalid ID supplied
    security:
      - api_key: []
  get:
    tags:
      - contact
    summary: Get contacts
    description: get all contacts
    operationId: getContacts
    produces:
      - application/json
    responses:
      "200":
        description: successful operation
        schema:
          type: array
          items:
            $ref: "#/definitions/Contact"
      "400":
        description: Invalid status value
    security:
      - api_key: []
/contact/{id}:
  get:
    tags:
      - pet
    summary: Get a contact
    description: Find a specific contact by its Id
    operationId: getContactById
    produces:
      - application/json
    parameters:
      - in: path
        name: id
        description: contact id
        required: true
        type: string
    responses:
      "200":
        description: successful operation
        schema:
          type: array
          items:
            $ref: "#/definitions/Contact"
      "400":
        description: Invalid status value
    security:
      - api_key: []
securityDefinitions:
  api_key:
    type: apiKey
    name: api_key
    in: header
definitions:
  Contact:
    properties:
      id:
        type: integer
        format: int64
      username:
        type: string
      firstName:
        type: string
      lastName:
        type: string
      email:
        type: string
      phone:
        type: string
```

La structure du document est relativement simple, et il est assez facile de la prendre en main. La première partie, « info », regroupe les informations générales sur le service : son nom, sa description, sa licence ou encore sa version. Les descriptions peuvent être enrichies grâce à la syntaxe markdown, ce qui permet de proposer un contenu agréable à lire pour les futurs utilisateurs de l'API. Ensuite, viennent les informations concernant l'hôte, le chemin de base des APIs et les protocoles supportés http et/ou https. Une fois ces informations décrites, nous pouvons rentrer dans le vif du sujet. Vous allez définir un ensemble de « paths » (chemins) et associer un ensemble de « verbes » http supportés pour ce dernier (get, delete, post, etc...). Pour chacun de ces appels possibles, vous pouvez définir les types de contenus acceptés et produits, les paramètres, qu'ils viennent du « body », de la « querystring » ou du « path ». « Security definition » permet de définir les différents moyens d'authentification utilisés/supportés par les différentes APIs (API key, OAuth etc...). Enfin, vous pouvez définir les schémas de données utilisés par l'API. Ainsi, le développeur saura exactement quels champs attendre et leurs types.

Génération de code

Swagger propose des outils de génération de code. Ils permettent de générer des squelettes serveur ou des libraires clientes pour consommer le service. Pour ce faire, deux approches sont possibles :

- La méthode longue : cloner swagger-codegen depuis github, le compiler puis l'utiliser en ligne de commande.
- La méthode simple : ouvrir votre fichier de définition sur « Swagger editor » et générer le code depuis le menu de l'éditeur. Il n'y a plus qu'à enregistrer le fichier généré.

Et voilà déjà quelques dizaines d'heures de développement qui sont économisées. Mais ce n'est pas tout ! Swagger UI, cette brique que l'on peut entre-autres installer avec Bower, propose une documentation et un « playground » très pratique. L'ensemble est généré dynamiquement à partir d'un fichier de spécifications Swagger. Il devient alors très aisé de tester les APIs sans écrire de code ou écrire des requêtes à la main avec un outil tel que Fiddler.

Travail avec l'existant

C'est le moment où vous vous dites : c'est bien beau, mais mon API est déjà développée, je fais quoi ? Pas de panique, d'une part, Swagger est suffisamment souple pour s'adapter à la plupart des schémas, d'autre part, il existe, pour un assez large spectre de technos, des frameworks permettant de générer les spécifications Swagger à partir du code et des commentaires de ce dernier. Par exemple, « Swagger.net » permet de documenter automatiquement des services développés avec Asp.net WebAPI. A titre d'exemple, il m'a fallu moins d'une heure pour mettre en place Swagger sur les APIs d'un backend d'app mobile .net préexistant.

Conclusion

Swagger vous accompagne de manière agnostique tant sur votre plateforme que sur la forme de votre API. Contrairement à d'autre approches, telles que ODATA, il est possible de mettre en place Swagger à tout moment sans ré-architecturer l'ensemble. Enfin, vous n'êtes probablement qu'à quelques dizaines de minutes d'une documentation dynamique efficace pour votre API, et de bibliothèques clientes pour la plupart des langages actuels. Alors, qu'attendez-vous pour vous lancer ? ⊞

Quels futurs pour les langages de programmation ?

Il y a un an, Programmez ! avait proposé un premier dossier complet sur ce sujet (n° 173) : débat autour de l'objet (utile ou pas utile ?), C++ 11, PHP, C# 6 /VB.Net 13, métaprogrammation, Java. Douze mois plus tard, est-ce que beaucoup de choses ont changé ?

Le monde des langages n'est pas inerte. C'est un monde vivant qui évolue régulièrement. Pour un non-initié, il croira qu'un langage est un langage et que rien ne change ou presque. Il saura sans doute que les langages possèdent des versions (Java 8, PHP 5, C# 5, etc.), mais sans forcément aller chercher plus loin...

Mi-2014, Apple crée la surprise en sortant de sa pomme, un nouveau langage : SWIFT. Neuf mois plus tard, SWIFT est déjà en version 1.2. Cette « rapidité » d'évolution s'explique pour plusieurs raisons : jeunesse du langage, bugs, rajout de nouvelles fonctions, amélioration des performances et de la stabilité, etc. Car finalement, lancer un langage n'est pas trop compliqué, mais le rendre utilisable pour des millions de développeurs le plus rapidement possible est un défi colossal. Et si SWIFT est l'avenir de la programmation sur iOS et OS X, Objective-C, vénérable langage de 30 ans, restera pour longtemps la référence.

Mais imposer un nouveau langage n'a rien de simple ! Demandez à Google ce qu'il en pense pour Dart et Go... Il y a quelques mois, nous évoquions un nouveau langage très orienté, OForth. Régulièrement, des langages très spécialisés apparaissent même si 99,99 % l'ignorent. Souvent, ces langages ne sortent pas de quelques laboratoires.

Dans ce dossier, nous allons faire des focus sur C++, Java, C#, JavaScript, HTML, PHP... mais nous verrons aussi que des paradigmes trop souvent ignorés par le développeur peuvent aujourd'hui être un formidable tremplin pour blinder son code, le rendre plus souple et plus lisible. Mais ils nécessitent une remise en question de ses méthodes de programmation, ce qui n'est pas facile à accepter.

Enjoy !

La rédaction

C++ 11 et la révolution du « modern C++ »

Depuis que C++ existe, le type vector<T> est le type roi de la librairie standard STL (Standard Template Library). Depuis le standard C++ 11, il faut se forcer à utiliser les smart pointers, car ils permettent de faciliter la gestion de la mémoire dynamique et surtout parce qu'ils permettent d'éviter les new et les delete.

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

Le type vector de la librairie standard STL

Un vector est une collection d'objets de même type. Chaque objet de la collection possède un index qui permet d'accéder à cet objet. Un vector est souvent nommé un « container » parce qu'il contient d'autres objets. Pour utiliser un vector, il faut faire un #include approprié. Dans nos exemples, on part du principe qu'un using std::vector est réalisé.

```
#include <vector>
using std::vector;
```

Un vecteur est une classe template. C++ possède à la fois des classes templates, et des fonctions templates. Les templates ne sont pas des classes ou des fonctions. C'est une déclaration pour le compilateur pour qu'il génère les classes ou les fonctions. C'est la phase d'instanciation. Pour une classe template, il faut lui spécifier le type à gérer via une série de <T>. Dans le cas du vector, voici comment le déclarer :

```
vector<int> ivec;          // vector de int
vector<Sales_item> Sales_vec; // vector d'objets Sales_items
vector<vector<string>> file;  // vector de vector de string
```

Dans cet exemple, le compilateur génère 3 types distincts : vector<int>, vector<Sales_item>, and vector<vector<string>>.
On peut définir des vectors pour contenir n'importe quel type de données ou presque. Il est possible d'avoir un vector de vector.
Le C++ 11 autorise à supprimer l'espace que l'on écrivait avant comme vector<vector<int> > et permet cela : vector<vector<int>>.

Initialisation du vector

On peut définir un vector vide comme suit :
vector<string> svec; // initialisation par défaut; svec n'a aucun élément.
Il est possible d'initialiser un vector avec des valeurs. Il est possible de copier des éléments:

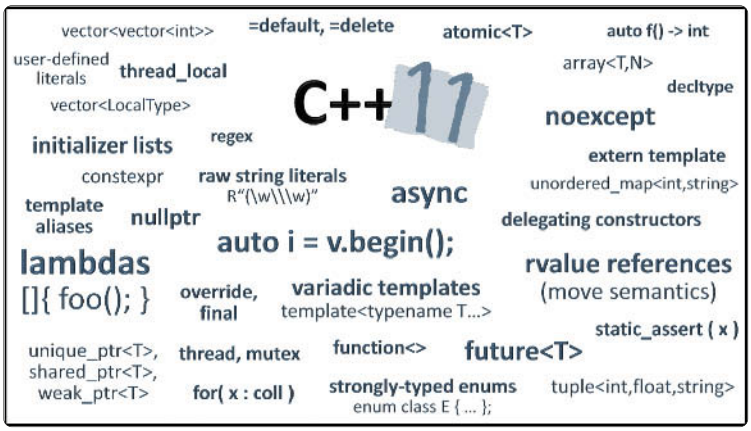
```
vector<int> ivec;          // vide
vector<int> ivec2(ivec);    // copie d'éléments de ivec dans ivec2
vector<int> ivec3 = ivec;   // copie d'éléments de ivec dans ivec3
vector<string> svec(ivec2); // erreur: svec contient des strings, pas des ints.
```

Avec le nouveau standard C++ 11, il est possible d'initialiser un vector avec une liste :

```
vector<string> articles = {"a", "an", "the"};
```

Le vector possède 3 éléments : a, an, the.

```
vector<string> v1{"a", "an", "the"}; // initialisation par liste
vector<string> v2("a", "an", "the"); // erreur
```



Autres initialisations

Il est possible d'initialiser un vector avec un compteur et une valeur de remplissage.

```
vector<int> ivec(10, -1);      // 10 éléments int, chacun initialisé à -1
vector<string> svec(10, "hi!"); // 10 strings; chacune initialisé à "hi"
```

Il est possible de ne donner que la taille de vector et d'oublier la valeur par défaut. Si le vector contient des int, ils seront initialisés à 0. Les éléments string seront automatiquement initialisés.

```
vector<int> ivec(10);   // 10 éléments, chacun initialisé à 0
vector<string> svec(10); // 10 éléments, chacun initialisé avec une chaine vide
```

Ajouter des éléments à un vector

Comment ajouter les éléments de 0 à 99 ? Il faut utiliser le membre push_back du vector. L'opération push_back prend une valeur et push cette valeur comme le dernier élément du vector.

```
vector<int> v2;      // vector vide
for (int i = 0; i != 100; ++i)
   v2.push_back(i);   // ajout de int à v2
// v2 contient 100 éléments, valeurs de 0..99.
```

Malgré le fait que l'on sache que l'on aura 100 éléments, nous déclarons v2 comme vide. Le standard requiert que les implémentations de vector puissent ajouter des éléments efficacement au run-time.

D'autres opérations sur vector

Opération	Explication
v.empty()	Retourne true si v est vide. Sinon retourne false.
v.size()	Retourne le nombre d'éléments dans v.
v.push_back(t)	Ajoute un élément de valeur t à la fin de v.
v[n]	Retourne une référence vers l'élément en position n dans v.
v1=v2	Remplace les éléments dans v1 avec une copie des éléments dans v2.
v1={a, b, c...}	Remplace les éléments dans v1 avec une copie des éléments de la liste.
v1==v2	v1 et v2 sont égaux s'il y a le même nombre d'éléments et de valeurs.
v1!=v2	opposé de v1==v2
<, <=, >, >=	Suivant l'ordre des valeurs, retourne un bool

Nous avons accès aux éléments d'un vector de la même manière que nous avons accès aux caractères d'une string : via leur position dans le vector. Il est possible par exemple d'utiliser un range-for pour traiter les éléments d'un vector :

```
vector<int> v{1,2,3,4,5,6,7,8,9};
for (auto &i : v)    // pour chaque élément dans v (i est une référence)
  i *= i;
for (auto i : v)     // pour chaque élément dans v
  cout << i << " "; // affiche l'élément
cout << endl;
```

Les membres empty et size correspondent, comme dans une string. Empty retourne un booléen qui indique si le vector possède des éléments, et size retourne le nombre d'éléments dans le vector :

```
vector<int>::size_type // ok
vector::size_type     // erreur
```

Utilisation de []

On peut utiliser l'opérateur [] pour fetcher des éléments qui existent :

```
vector<int> ivec;     // vector vide
cout << ivec[0];      // erreur: ivec n'a aucun élément

vector<int> ivec2(10); // vector avec 10 éléments
cout << ivec2[10];    // erreur: ivec2 has les éléments 0 . . . 9
```

C'est une erreur de fetcher un élément qui n'existe pas.
Le meilleur moyen de ne pas échouer est d'utiliser un range le plus souvent possible.

Parcours du vector

Un vecteur peut être parcouru via un itérateur. Tous les containers de la librairie standard exposent des itérateurs. Comme les pointeurs, les itérateurs donnent accès à un objet de manière indirecte. Les membres donnent accès aux itérateurs dont begin() et end(). Il existe aussi des membres cbegin() et cend() pour avoir un itérateur const. Il est même possible d'avoir un itérateur reverse via rbegin() et rend() et crbegin() et crend() ;

```
auto it = v.begin() ; // auto permet de masquer le type de l'itérateur.
```

Un itérateur se manipule avec l'opérateur d'incrément ++. L'accès à la valeur mémoire se fait avec l'utilisation de *. Dans le cas du vecteur de string, l'itérateur s'écrit vector<string> ::iterator it ;

```
vector<string> kids;
kids.push_back("Edith");
kids.push_back("Lisa");
kids.push_back("Maggie");

for (auto &it1 : kids)
{
    cout << it1 << endl;
}

for (vector<string>::const_iterator it2 = kids.cbegin(); it2 != kids.cend(); ++it2)
{
    cout << *it2 << endl;
}

for (auto it3 = begin(kids); it3 != end(kids); ++it3)
{
    cout << *it3 << endl;
}
```

Il existe une convention depuis C++ 11 qui consiste à faire begin(kids) au lieu de kids.begin() ;

Introduction à la mémoire dynamique

Les programmes que l'on écrit utilisent des objets qui ont une durée bien définie. Les objets globaux sont alloués au démarrage de l'application et détruits quand l'application se termine. Les objets locaux sont créés et détruits au début du bloc, puis à la fin de celui-ci. Les objets locaux static sont alloués avant leur première utilisation, et détruits quand le programme se termine.
En plus de supporter les objets automatiques et static, C++ nous permet d'allouer des objets dynamiquement. Ces objets dynamiquement alloués ont un cycle de vie qui est indépendant de quand ils ont été créés ; ils existent jusqu'à ce qu'ils soient explicitement libérés.
Libérer proprement les objets dynamiques tourne souvent à alimenter les sources d'erreurs. Pour rendre l'allocation dynamique d'objets plus sûre, la librairie définit deux smart pointers qui gèrent dynamiquement les objets alloués. Les smart pointers permettent la libération automatique de ces objets, et ce, de manière appropriée.
Nos applications ont utilisé seulement de la mémoire static ou mémoire stack (sur la pile). La mémoire static est utilisée pour les objets static, pour les membres static de classe, et pour les variables définies en dehors des fonctions. La mémoire stack (sur la pile) est utilisée pour les objets non static définis dans les fonctions. Les objets alloués en static ou en mémoire stack sont automatiquement créés et détruits par le compilateur. Les objets stack existent seulement dans le bloc dans lequel ils sont définis et qui s'exécute ; les objets static sont alloués avant d'être utilisés, et sont détruits quand l'application se termine.
En plus de la mémoire static ou stack, une application peut utiliser un pool de mémoire. Cette mémoire est appelée free store ou heap (le tas). Les applications utilisent le heap pour les objets qui sont alloués dynamiquement – cela veut dire pour les objets que l'application alloue au runtime. L'application contrôle le cycle de vie des objets dynamiques mais le code doit explicitement détruire les objets lorsqu'on n'en a plus besoin. C'est parfois délicat (tricky J).

Mémoire dynamique et smart pointers

En C++, la mémoire dynamique est gérée au travers des opérateurs new et delete. L'opérateur new alloue, et optionnellement initialise, un objet en mémoire et retourne un pointeur sur cet objet. L'opérateur delete, qui prend un pointeur sur un objet dynamique, détruit l'objet et libère la mémoire associée. La gestion de la mémoire dynamique est touchy car il faut libérer toute la mémoire allouée sinon vous avez un memory leak. L'autre erreur est de posséder des pointeurs sur une zone de mémoire non valide, et là, c'est plantage de l'application dès que le pointeur cherche à être utilisé. Pour rendre plus facile la gestion de mémoire dynamique, la nouvelle librairie standard fournit 2 smart pointers qui gèrent les objets dynamiques. Un smart pointer est comme un pointeur normal avec une différence notable qu'il libère automatiquement l'objet sur lequel il pointe. Il y a deux types de smart pointers qui diffèrent dans leur manière de gérer le cycle de vie de l'objet. Le shared_ptr<T> permet

que plusieurs pointeurs ciblent le même objet, et unique_ptr<T> qui possède l'objet sur lequel il pointe. Il existe aussi weak_ptr<T> qui est une référence faible sur un objet géré via shared_ptr<T>. Tous ces smart pointers sont des templates définis dans le fichier d'entête memory.

Opérations communes à shared_ptr et unique_ptr

Opération	Explication
shared_ptr<T> sp	Smart pointer null qui pointe sur un objet T
unique_ptr<T> up	Smart pointer null qui pointe sur un objet T
p	Utilise p comme une condition; true si p pointe sur un objet
*p	Déréférence p pour obtenir l'objet sur lequel p pointe
p->mem	Synonyme pour (*p).mem
p.get()	Retourne le pointeur dans p.
swap(p,q)	Swap les pointeurs dans p et q
p.swap(q)	Swap les pointeurs dans p et q

Opérations spécifiques à shared_ptr

Opération	Explication
make_shared<T>(args)	Retourne un shared_ptr sur la mémoire allouée et initialise l'objet via args
shared_ptr<T> p(q)	p est une copie de q. Incrémente le compteur de référence interne
p=q	Incrémente le compteur de référence de q
p.use_count()	Retourne le nombre d'objets partagés avec p
p.unique()	Retourne true si p.use_count vaut 1 sinon false

La fonction make_shared<T>()

Le moyen le plus sûr pour allouer et utiliser de la mémoire dynamique est d'appeler la fonction make_shared de la librairie. Cette fonction alloue et initialise l'objet en mémoire dynamique et retourne un shared_ptr qui pointe sur l'objet. Lorsque l'on appelle makes_shared, il faut spécifier le type d'objet que l'on veut créer. On fait comme avec une classe template sauf que c'est une fonction.

```
// shared_ptr pointe sur un int qui vaut 42
shared_ptr<int> p3 = make_shared<int>(42);
// p4 pointe sur une string de valeur 9999999999
shared_ptr<string> p4 = make_shared<string>(10, '9');
// p5 pointe sur un int qui a une valeur initialize à 0
shared_ptr<int> p5 = make_shared<int>();
```

Il est possible de passer des arguments à make_shared<T>(args) pour faire un appel désiré au constructeur de l'objet. L'utilisation de auto rend aussi le code plus lisible.

```
// p6 pointe sur un vector<string> dynamiquement alloué et vide
auto p6 = make_shared<vector<string>>();
```

Destruction automatique des objets

Quand le dernier shared_ptr qui pointe sur un objet est détruit, la classe shared_ptr détruit automatiquement l'objet sur lequel il pointe. Ceci est réalisé via le destructeur de l'objet. Le destructeur libère la ressource que l'objet a alloué. Dans le cas du vector, il y a plusieurs opérations d'allocation mémoire pour gérer les objets du vector (le vector grossit automatiquement). Le destructeur du shared_ptr décrémente son compteur de référence interne. Dès qu'il arrive à 0, le destructeur détruit l'objet pointé par la shared_ptr et libère la mémoire utilisée par cet objet.

Ne pas mixer les pointeurs ordinaires et les shared_ptr

Un shared_ptr peut faire sa destruction seulement avec des shared_ptr qui sont passés en copie à lui-même. C'est pour cela qu'il faut utiliser make_shared (qui renvoie un shared_ptr) plutôt que new. De telle manière, on associe un shared_ptr à l'objet en même temps que son allocation. Considérons le code suivant :

```
// ptr is créé et initialize lorsque la function est appellée
void process(shared_ptr<int> ptr)
{
   // use ptr
} // ptr sort du scope et est détruit
```

Le paramètre de la fonction process est passé par valeur, donc l'argument de process est copié dans ptr. Copier un shared_ptr incrémente son compteur de référence. De ce fait, à l'intérieur de process le compteur est au moins à 2. Quand process se termine, le compteur de référence de ptr est décrémenté mais ne peut pas aller à 0. La variable locale ptr est détruite mais la mémoire sur laquelle ptr pointe ne sera pas supprimée. La bonne méthode pour utiliser cette fonction est de passer un shared_ptr.

```
shared_ptr<int> p(new int(42)); // reference count vaut 1
process(p); // copier p incrémente le compteur ; vaut 2 dans process
int i = *p; // ok: reference count vaut 1
```

Il est dangereux d'utiliser un pointeur standard pour accéder à un objet possédé par un smart pointer, parce que nous ne savons pas quand l'objet sera détruit et sa mémoire libérée.

Le C++ Moderne

La standard C++ ISO version C++11 introduit de nouvelles fonctionnalités qui bouleversent notre façon d'écrire du C++. Les lambdas, auto, les range-for, les shared_ptr<T>, les lists initializers, et tant d'autres fonctionnalités font que le code nouveau doit adopter ces nouvelles conventions et idiomes. Par contre, que devons-nous faire du code legacy. La réponse est simple : seul le nouveau code doit être écrit en C++ moderne. Le standard évolue en C++14 et C++17 mais il n'y aura pas de grandes révolutions comparées à tout ce que apporte C++11. C++11 était la renaissance de C++. C'est une nouvelle façon d'écrire, une nouvelle façon de penser C++. Voici un rappel de tout ce qui est introduit dans C++11 ; ce schéma est fourni par Herb Sutter, chairman du comité ISO C++ (schéma du début).

Conclusion

La mémoire se manipule facilement et grâce aux smart pointers, l'écriture des applications est très simple. Qui sait qu'il est possible de faire des applications sans new et delete. Maintenant vous savez ! Et pourtant, on n'a pas besoin de garbage collector comme en .NET. Les objets qui se libèrent automatiquement dans d'autres langages comme C# tirent parti d'autres systèmes pour gérer le cycle de vie des objets via différents niveaux (génération de GC). En C++, la règle est très simple. Il existe un compteur interne pour les références, et dès que celui-ci tombe à 0, POUM ! L'objet est shooté et la mémoire est libérée. Le message de cet article est simple : utilisez le container vector et les smart pointers ; c'est simple et efficace, et bug-free. Les warriors du C++ connaissent cela avec Boost depuis 2001. Les diverses implémentations du compilateur C++ de Microsoft sont toujours en retard par rapport à GCC, mais maintenant la situation est bonne. Bref, utilisez les conventions du C++ 11 et vous entrerez dans le monde du C++ moderne.

Java : plateforme du passé, du présent et encore du futur ?

Lancée officiellement en 1996, la plateforme Java s'est rapidement imposée comme la solution de référence en entreprise pour réaliser des applications aussi bien client lourd que client léger. L'essor d'Internet couplé aux possibilités de la plateforme en la matière lui ayant largement permis de s'imposer. Toujours bien installée en entreprise, la plateforme Java doit désormais se transformer afin de répondre aux nouvelles problématiques du monde informatique. De la réussite de ce virage stratégique dépendra fortement le futur de la plateforme. Tour d'horizon des défis auxquels Java devra faire face à l'avenir.

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

Venant tout juste de fêter les 20 ans de sa création dans les laboratoires de Sun, la plateforme Java continue d'être considérée comme la solution numéro 1 de référence pour la réalisation d'applications d'entreprise. Cette tendance forte ne s'est jamais vraiment démentie depuis la création de la plateforme, et Java demeure toujours leader aussi bien côté client lourd que côté client léger. Néanmoins, si la plateforme Java demeure si prisée, elle le doit également à son ouverture et au fait qu'elle sert de plateforme d'exécution à des langages de plus en plus populaires tels que Groovy ou Scala, pour ne citer qu'eux. Le fort essor connu par ces langages a mis de fait au grand jour une des faiblesses majeures de la plateforme Java : son langage éponyme n'évolue pas assez vite et souffre d'un nombre important de limitations qui viennent freiner la productivité des développeurs.

Etat des lieux en 2015

La version 8 de Java est finalement arrivée début 2014 en apportant son lot de nouveautés avec pour but premier d'améliorer la productivité des développeurs. Ainsi, les Lambdas expressions mettent à la portée du plus grand nombre la programmation fonctionnelle. Combinées aux Streams, elles permettent des constructions puissantes. En outre, il est bon de souligner que la possibilité de laisser le soin à la JVM d'exécuter les traitements effectués par les Streams de manière parallèle est un plus indéniable pour démocratiser l'emploi du multithreading sur la plateforme Java. Le gros travail réalisé pour améliorer les Collections avec un support complet des Lambdas expressions est également à mettre au crédit des architectes de Java. Les interfaces fonctionnelles ainsi que la notion de méthode virtuelle d'extension qui découlent de ces précédentes modifications viennent combler plusieurs manques au sein du langage. Enfin, il faut souligner l'important travail fait autour de Java FX afin de proposer une solution alternative de qualité pour remplacer le framework Swing dans la réalisation d'applications Java client lourd. Les développeurs Java ne s'y sont d'ailleurs pas trompés, adoptant massivement la plateforme pour bénéficier au plus vite de ces ajouts qui profitent également aux langages basés sur la plateforme Java. La version 8 du JDK aura également été l'occasion d'un travail en profondeur pour améliorer la sécurité de la JVM, conduisant du même coup Oracle à repousser une fois de plus le projet de modularisation du JDK. Le désormais tristement célèbre projet Jigsaw est scopé pour Java 9 attendu au début de l'année 2016, mais nous y reviendrons par la suite. Côté Web, la spécification Java EE a connu de nombreux ajustements depuis sa création et su, depuis sa version 5, se remettre en question afin de tirer pleinement parti des meilleures pratiques recensées au sein de l'écosystème Java. C'est ainsi qu'un grand nombre des principes structurants du framework Spring ont été directement intégrés au sein de Java EE 6. Il en avait été de même précédemment avec les principes issus du framework ERM Hibernate à l'origine de la spécification JPA notamment. La version 7 de Java EE s'est quant à elle concentrée pleinement sur le Web avec un support étendu de la nouvelle spécification HTML 5 pour en simplifier l'utilisation et améliorer la productivité des développeurs. Au rang des autres nouveautés majeures, on peut également citer l'introduction d'une API pour faciliter la manipulation du format JSON ou encore l'ajout d'une API dédiée aux applications Batch via la JSR 352. Là encore, Oracle a su tirer pleinement parti des concepts issus du framework Spring Batch afin de les intégrer directement au sein de la spécification. Enfin, la version 7 aura également été l'occasion de préparer le support pour les environnements Cloud et le PAAS notamment qui sera un des principaux objectifs visés par Java EE 8. Pour ce faire, des améliorations ont dû être apportées concernant la définition des ressources, la configuration de la sécurité ou encore la définition des schémas de bases de données.

Modularisation du JDK

Attendu pour le début d'année 2016, Java 9 doit être la première version du JDK modulaire avec l'apparition du tant attendu projet Jigsaw qui doit permettre aux développeurs de la plateforme Java de réaliser des applications n'ayant pas besoin d'un JDK entier pour s'exécuter si elles n'utilisent qu'un sous-ensemble du JDK limité à un profil Compact donné. Un module Jigsaw est ainsi une collection de classes Java, de bibliothèques natives ou d'autres ressources accompagnées de certaines métadonnées. Compte tenu de l'ampleur de la tâche, le projet Jigsaw est découpé au sein d'une JSR (Java Specification Request) et de 3 JEPs (JDK Enhancements Proposals) :

- La JSR 376 qui est le composant central du projet Jigsaw, définit le système de module pour la plateforme Java,
- La JEP 200 vise à définir une structure modulaire pour le JDK. Il s'agit de la proposition chapeau pour l'ensemble du projet,
- La JEP 201 définit la réorganisation du code source du JDK en modules,
- La JEP 220 s'intéresse quant à elle au runtime du JDK et du JRE qu'elle vise à découper en modules.

Cette modularisation du JDK permettra la suppression des fichiers associés au monilithique JAR du runtime Java nommé rt.jar. Les différents profils Compact introduits au sein de Java 8 servant de référence pour le découpage du JDK en sous-ensembles. Le projet Jigsaw s'inscrit clairement dans la volonté d'optimiser les performances de la JVM et de l'alléger pour son exécution au sein du Cloud.

En installant les derniers builds du JDK 9, il est d'ores et déjà possible de tester la création et l'utilisation de modules Jigsaw. Considérons un module com.ssaurel en version 0.1 contenant une classe Hello. Ce module requiert l'emploi d'un autre module nommé org.tools en version 1.1. La déclaration du module se ferait comme suit au sein d'un fichier module-info.java :

```
module com.ssaurel @ 0.1 {
  requires org.tools @ 1.1;
  class com.ssaurel.Hello;
}
```

Au sein de cette déclaration, puisqu'aucun module de la plateforme n'a été ciblé, le JRE complet sera utilisé ce qui reviendrait à la déclaration suivante :

```
module com.ssaurel @ 0.1 {
  requires jdk.base; // version la plus haute
  requires org.tools @ 1.1;
  class com.ssaurel.Hello;
}
```

Pour pouvoir compiler ce module, il faudra ainsi définir la hiérarchie de répertoires suivante :

```
src/classes/com/ssaurel/Hello.java
src/classes/module-info.java
```

La compilation du module se faisant ensuite via la ligne de commande suivante :

```
javac -d modules -modulepath modules -sourcepath src `find src -name '*.java'`
```

Il reste ensuite à installer le module au sein d'une bibliothèque de modules que l'on nommera ici maLib :

```
jmod -L maLib install modules org.tools com.ssaurel
```

On notera ici le recours au nouvel exécutable jmod ajouté au sein du JDK 9. Enfin, l'exécution du module maLib pourra être réalisée en lançant la ligne commande suivante :

```
java -L maLib -m com.ssaurel
```

L'option -m permet ici de préciser que l'on souhaite exécuter une application en mode module. La classe Hello du module est alors exécutée. Si cet exemple se veut volontairement simpliste, les possibilités offertes par Jigsaw vont bien plus loin et permettront à la plateforme Java de bénéficier enfin d'un véritable cadre d'exécution modulaire.

De nouveaux enjeux pour le JDK

La modularisation du JDK est un enjeu crucial pour permettre à la plateforme de répondre aux nouveaux enjeux de l'informatique d'entreprise. Elle doit faciliter le build, le déploiement ainsi que l'exécution d'applications Java en mode Cloud. Au-delà de Java 9, les enjeux de la plateforme vont bien plus loin. Les architectes de la plateforme ont ainsi mis en avant un certain nombre de points pénalisant fortement la JVM dont certains sont d'ores et déjà résolus ou en passe de l'être :

- Nommage des méthodes Java qui doivent être adressées par la JSR 202,
- Difficultés liées au mode d'invocation des méthodes qui sont adressées par la JSR 292 et le nouveau mode d'invokedynamic,
- La définition des types qui pose des problèmes de chargement des classes qui doit être adressées via la spécialisation et les types valeurs. Le projet sous-jacent à cette problématique étant le projet Valhalla,
- Le temps de chargement actuel de la JVM qui pénalise le temps de démarrage des applications et l'empreinte mémoire. Cette problématique doit être adressée par le fameux projet Jigsaw et par des améliorations au niveau de la compilation AOT,
- La Concurrence laisse à désirer avec un modèle de threads trop complexe à maîtriser. Cette problématique est en partie adressée grâce à l'introduction des Streams de Java 8 et en cours d'étude plus poussée au sein du projet Sumatra,
- La mutabilité qui doit être améliorée avec les types valeur,
- Un travail sur les types de données doit être réalisé pour supporter le 64 bits sur tous les types, ce qui se traduit notamment par le projet lié aux Arrays 2.0,
- Amélioration de l'interopérabilité avec les bibliothèques de code natives puisque JNI est limité et dépassé. Les réflexions sur cette problématique sont regroupées au sein du projet Panama.

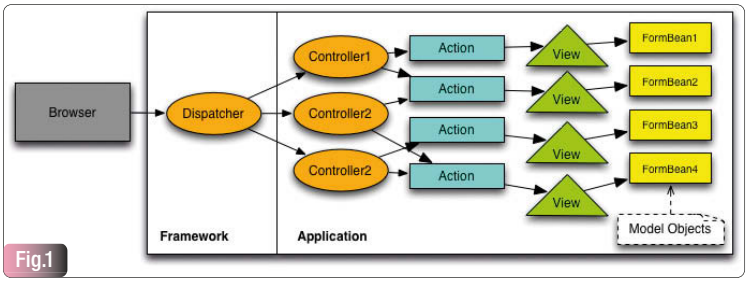


Fig.1

Architecture du framework MVC 1.0

Pour faire face à ces points sensibles, les développements autour de la JVM et du JDK au cours des 15 prochaines années seront centrés autour de ces 8 enjeux principaux :

- Proposer un modèle de mémoire uniforme aussi bien pour les objets que pour les tableaux ou les types primitifs,
- Proposer un modèle de mémoire efficace,
- Optimiser le code partagé,
- Simplifier la complexité liée à l'utilisation du code multithread,
- Intégration robuste avec le code des langages non-managés (C/C++) pour favoriser l'interopérabilité avec Java,
- Rendre l'intégration avec des langages basés sur la plateforme Java plus sûre et fiable,
- Assurer la compatibilité avec les anciennes versions du JDK,
- Garantir un niveau de performance maximal en tirant parti des avancées matérielles des ordinateurs.

On le voit clairement les enjeux pour construire une plateforme Java performante et répondant aux nouvelles problématiques d'entreprise à l'orée de l'an 2030 sont nombreuses et complexes. Nommés bien souvent avec des noms exotiques, les projets s'y afférant sont ambitieux, et il faut espérer qu'ils auront une évolution plus rapide que le projet Jigsaw qui sera la première étape au Java de demain et devrait arriver dès 2016.

Java EE 8

Parallèlement à l'évolution de la plateforme Java, la spécification Java EE évolue pour s'adapter aux besoins des entreprises. Attendue pour fin 2016, Java EE 8 a pour ambition de continuer le travail d'intégration avec le Cloud entrepris dans Java EE 7. Outre ce travail de fond, cette nouvelle mouture doit apporter au développeur une API de binding JSON, une évolution majeure de l'API Servlet avec un passage en 4.0, une révision de JSF en version 2.3 ainsi que la définition d'une API de Sécurité pour Java EE qui se doit de renforcer ce domaine crucial pour l'intégration des applications Java d'entreprise dans le Cloud.

Parallèlement au support de JSF qui continuera dans Java EE 8, un framework Web orienté action sera ajouté à la spécification via le projet MVC 1.0. L'architecture de ce dernier (figure 1) n'étant pas sans rappeler les frameworks populaires Apache Struts ou Spring MVC Fig.1.
Bien que l'ajout de MVC 1.0 ne remette pas en cause le support d'Oracle pour le framework JSF, il est bon de se demander si le rajout d'un autre framework pour la gestion des IHM au sein de Java EE ne viendra pas complexifier l'ensemble et dérouter les nouveaux entrants sur la plateforme.

Effort pour Java dans l'embarqué

Distancé, et de loin, par Google et son OS Android dans la guerre de l'embarqué au niveau des smartphones et des tablettes, Oracle continue toujours de pousser Java ME en prônant un rapprochement avec une version embarquée de Java SE. Le travail autour de la modularisation du JDK devant ici servir à proposer une version de Java SE allégée pour répondre aux contraintes de l'embarqué. La sortie de cette version est attendue également pour début 2016 et devrait être alignée sur celle de Java 9 si tout se déroule comme prévu. Dans ce domaine, il semble que l'avance prise par Google soit irrattrapable en l'état pour Oracle, et il est fort à parier que l'avenir du langage Java dans l'embarqué passe plutôt par le géant de Mountain View.

Conclusion

Plateforme du passé mais également du présent, Java va devoir relever un certain nombre de défis pour demeurer celle du futur. La prise en main d'Oracle ne se sera finalement pas faite sans soucis puisque la plateforme aura connu plusieurs coups d'arrêt avec différents reports et retards au niveau de Java 7 ou de Java 8 notamment. Le projet Jigsaw étant l'exemple même des difficultés d'évolution que connaît la plateforme du fait de sa taille importante et des problèmes que la compatibilité ascendante engendre. Dans ce contexte, la sortie de Java 8 avec l'apparition des Lambdas expressions et des Streams ainsi que les importants correctifs réalisés au niveau de la sécurité sont des signaux forts envoyés à la communauté. Oracle a désormais repris les commandes du navire Java et l'avenir, bien que semé d'embuches, s'annonce radieux pour la plateforme !

L'agenda 2015-2016 des langages

Les langages évoluent régulièrement, plus ou moins rapidement. Ce dossier vous l'a démontré. Faisons le point sur quelques autres langages que nous n'avons pas abordés.

Go 1.5 : mai-juin 2015

Go est un langage initié par Google. Il est en version 1.4 depuis décembre 2014. Si le rythme des versions est conservé, Go 1.5 apparaîtra vers le mois de mai ou juin. Le repository du projet sera sur Git et non Mercurial. Le ramasse-miettes sera concurrent avec une latence réduite. Le plus gros changement sera que le C sera totalement exclu du langage, du compilateur, du linker, etc. Le support 64-bits sera étendu sur PowerPC, un meilleur support d'Android et sans doute des ARM 64.

Python 3.5 : septembre 2015

La fondation Python travaille actuellement sur la version 3.5 du langage Python (actuellement en v3.4.x). Les premières versions alphas sont disponibles depuis février. La version finale est attendue pour mi-septembre. A partir de la bêta 1 (attendue vers le 24 mai), les fonctionnalités seront gelées, c'est à dire, que la liste sera considérée comme complète. La version 3.5 doit introduire de nombreuses nouveautés et améliorations : lanceur Python en environnement virtualisé, amélioration des séquences de démarrage, ajout du « % » dans bytes et bytearray, nouveau module zipapp, changements dans les API C (nouvelles fonctions calloc), dépréciations de certaines fonctions / modules / API, changements dans les API Python.
Site : https://docs.python.org/dev/whatsnew/3.5.html

Ruby 2.3 : décembre 2015 (?)

Le langage est actuellement en version 2.2.x. La v2.1 avait été une importante mise à jour du langage (notamment sur le ramasse-miettes). La prochaine version, la 2.3, est prévue pour décembre prochain. Le développement a débuté il y a quelques mois. Il y a actuellement 117 demandes avec différentes priorités : division sur les nombres négatifs, ajout des ??a, support de programmation fonctionnelle, dépréciations de divers éléments... Des dizaines de demandes sont rejetées par les développeurs. On devrait connaître les nouveautés de cette 2.3 durant l'été.

Perl 6 : décembre 2015 (?)

Le langage Perl est un des langages historiques. Le langage est actuellement en version 5.x.x. Il est toujours beaucoup utilisé et la communauté est très riche et active. Perl 6 est en gestation depuis 2000.
Les équipes espèrent une première version développeur en septembre et la version finale en décembre mais ce planning est provisoire.

Scala 2.12 : janvier 2016

Scala est un langage fonctionnel de l'univers Java. Scala 2.12 nécessitera Java 8 obligatoirement. Cette version partagera des fonctionnalités avec la 2.11, d'autres seront exclusives à la version.
Cette version permet d'utiliser des interfaces fonctionnelles sans wrapper, utilisation de la librairie forkjoin (JDK) en remplacement de la librairie actuelle, intégration du SIP-20 (amélioration de l'initialisation des lazy val).
Le projet Scala a déjà annoncé deux autres versions : Aida et Giovanni. Ces versions se focaliseront sur le nettoyage de la syntaxe, les tailles et les performances des librairies, l'optimisation du compilateur.
Site : http://www.scala-lang.org/news/roadmap-next

HTML 5.1 : fin 2016 (?)

HTML 5 a mis de longues années à être totalement spécifié et disponible en version finale. Le W3C veut améliorer le processus et depuis plusieurs mois, une équipe travaille à définir et à spécifier les fonctions de la 5.1. Ce sera la 5e grande version de HTML. La 5.1 touchera peu le cœur et introduira de nouvelles fonctions. Par exemple, la balise <picture> sera plus souple que la balise <img>. L'objectif est de pouvoir faire du responsive design automatiquement ou tout du moins plus rapidement.
Site : http://www.w3.org/TR/html51/

Méta-programmation et AOP selon Aspectize

La science, l'industrie et l'informatique sont trois activités humaines qui exploitent et explorent les répétitions, les récurrences. Newton dit que toutes les chutes se ressemblent et que l'essence de cette ressemblance est résumée par la formule G. Ford décompose le processus de fabrication d'une voiture en une chaîne d'actions répétitives à exécuter les unes après les autres. Et Turing imagine la machine ultime, la machine qui sait automatiser tout ce qui est automatisable.

Frédéric Fadel
Aspectize

Cela fait plus de 50 ans qu'on parle aux ordinateurs à travers des programmes, pour faire faire au silicium nos tâches répétitives et sans valeur. L'écriture des programmes, les langages de programmation et les ruses et astuces du programmeur pour transformer les idées en produits logiciels ont peu - ou pas - évolué dans leur esprit depuis trente ans. Le quotidien du programmeur ressemble à celui de l'ouvrier spécialisé. Il écrit et réécrit le même code pour faire exécuter la même tâche au même silicium, alors que ce quotidien devrait ressembler plus à celui de Newton, Ford ou Turing qui ont imaginé et inventé de nouvelles façons de faire. L'effervescence en informatique s'explique partiellement par ce constat. Tout programmeur cherche d'une manière ou d'une autre à échapper à cette situation, d'où pléthore de styles, technologies, approches et langages qui se suivent et se ressemblent.

On peut néanmoins noter des tendances de fond. L'informatique évolue du rigide au souple, du statique au dynamique, du physique au logique, de l'impératif au déclaratif, du synchrone vers l'asynchrone, et du technique vers le métier. Les langages généralistes - tels qu'on les connaît - ont donné tout ce qu'ils pouvaient. Le prochain grand pas qui sera franchi par l'informatique ne se fera pas à travers un langage mais par une approche de programmation qui peut dès aujourd'hui se décliner avec bon nombre de langages classiques.

Parmi ces approches, nous allons explorer la méta-programmation, l'AOP et un mélange subtil des deux à travers l'approche d'Aspectize.

La méta-programmation

L'idée de la méta-programmation est vieille comme la machine de Turing. En effet la machine universelle de Turing est une machine (programme) capable de simuler toutes les autres. Elle prend en entrée un programme et elle produit comme sortie le résultat de l'exécution de ce programme. Le programme universel est un méta-programme, il « manipule » d'autres programmes. On utilise la méta-programmation pour se répéter moins, mais aussi pour être plus adaptable, plus dynamique, pour avoir moins d'instructions ou expressions **codées en dur** dans un programme.

La méta-programmation - du préprocesseur et macros du C à Roslyn, en passant par la réflexion (introspection) et l'émission de l'**IL** (*Intermediate Language*) à la volée en .net - est l'art pour un programme de s'auto analyser, s'auto changer ou produire un autre programme. Ou bien encore, la méta-programmation c'est traiter un programme ou une de ses parties comme une donnée. Cela peut se faire statiquement à la compile, c'est le cas des générateurs de code, ou, mieux encore, dynamiquement à l'exécution (comme pour certains « sérialiseurs » sous .net). La méta-programmation est une notion relative. Comparé à aligner une suite d'opérateurs NAND, écrire de l'assembleur c'est déjà de la méta-programmation. En passant au niveau méta, on perd le contrôle du détail insignifiant et on gagne le contrôle sur la fonction à accomplir. Ecrire du code asynchrone en C# aujourd'hui en utilisant les mots-clé **async** et **await** revient exactement à ça. On laisse le compilateur du C# s'occuper des détails de façon **déclarative**, alors que voilà quelques années on codait (souvent mal) ces détails sous forme **impérative**. Ainsi, programmer en .net nous éloigne du processeur mais nous « rapproche » du *but*.

Domain Specific Language (DSL) pour aller beaucoup plus loin

Le programmeur chevronné remarque les répétitions et récurrences dans son code, et utilise des outils et techniques de méta-programmation pour échapper au niveau de détails que lui impose son environnement ou langage de programmation. Il utilise alors un « langage » de plus haut niveau pour exprimer ses intentions : on parle alors de DSL. Ces DSL servent à résoudre le problème de répétitivité spécifique. Parfois l'usage de ces techniques est nouveau et cela effraie le développeur qui s'était accommodé de cette tâche répétitive ; d'autres fois cet usage est tellement répandu qu'il n'imagine même pas s'abaisser au niveau inférieur et accomplir la tâche comme avant. Typiquement on utilise volontiers un éditeur de formulaire, pour ne pas avoir à s'occuper - à travers l'écriture de code - des position, taille et comportement standards des contrôles. On préfère s'en occuper d'une manière plus WYSIWYG en faisant générer le code répétitif par un outil, et on contrôle l'outil d'une manière plus méta, plus proche de ce qu'on veut produire. Pour pouvoir tirer profit de la méta-programmation, il faut d'abord identifier le domaine (la tâche) qui est récurrent et sans valeur, et disposer d'un langage ou d'un outil permettant d'exprimer la tâche avec des termes ou actions de plus haut niveau que l'alignement d'instructions exécutables.

La méta-programmation est notamment utilisée dans le domaine de l'accès aux données à travers SQL, puisque beaucoup d'opérations sont complètement prévisibles, génériques et automatisables, comme la mise à jour, la création ou suppression d'une entité. La liaison de données ou *data binding* est un autre domaine qui exploite la méta-programmation à travers l'introspection dans .net. On se répète moins si on identifie et isole la récurrence. Ce n'est pas toujours facile tant les aspects techniques de la programmation, des langages, des systèmes de communication et de stockage sont emmêlés dans le code avec le but à atteindre : **écrire un logiciel qui rend service à des humains**.

Aspect Oriented Programming pour éviter les mélanges

L'Aspect Oriented Programming ou AOP, c'est l'art de la séparation, ou plutôt du **non-mélange,** des aspects indépendants, non-mélange du quoi et du comment, non-mélange du code technique nécessaire et du code métier qui ne devrait pas être contaminé par le technique souvent prévisible. L'AOP comme beaucoup de techniques et idées intéressantes en informatique est née dans les labos du Xerox PARC. Elle a été mise en œuvre dans les produits MTS et COM+ de Microsoft entre 1996 et 2000 et finalement, elle est présente dans l'environnement .net comme une technique de programmation comme une autre. Quand on parle d'AOP on parle souvent d'inversion de contrôle ou **principe de Hollywood** : « Don't call us, we'll call you ». En effet, contrairement à l'usage des librairies où l'utilisateur d'un service doit le connaître pour l'appeler, en AOP on écrit du code qu'on n'appelle jamais. On laisse le soin des appels à l'environnement ambiant. Dans .net on décore le code métier par des « attributs » et l'environnement .net (*design time* ou

runtime) intervient pour agir à l'endroit choisi par le développeur. De tels attributs existent nativement pour traiter les aspects liés par exemple à la sécurité ou aux détails de sérialisation. Le principe de Hollywood permet à un environnement d'exécution de fournir de nombreux services techniques sans que le code métier ait même connaissance des détails d'implémentation et d'usage de ces services techniques. Cette séparation entre code technique et code métier est un catalyseur de réutilisation du code technique et de clarté du code métier. On peut toujours améliorer la concision de son code avec la méta-programmation ou l'AOP. Mais la mise en place de ces techniques de façon industrielle et systématique est un travail à plein temps en soi. Le développeur métier ne devrait pas s'en soucier.

L'approche MVS (Modèle, Vue, Service) d'Aspectize pour être souple et dynamique

Chez Aspectize on exploite les machines pour la récurrence et on valorise le développeur pour ce qu'il est seul capable de faire : comprendre un besoin, le décomposer, concevoir, développer et livrer un résultat par petits bouts, avec le minimum de code nécessaire de telle sorte que chaque cycle ne dépasse pas plus de 2-3 heures Fig.1. La technologie d'Aspectize permet de développer du **Service as a Software** sous forme d'applications Web SPA (Single Page Application), sophistiquées et 100 % sur mesure. Elle exploite les HttpHandlers d'ASP.Net et les services d'Azure en utilisant deux DSL intégrés à Visual Studio : un DSL graphique, **Entity Designer**, pour définir le modèle des données qui servira pour le stockage, transport et affichage de l'information, et un DSL textuel, **Aspectize Binding Language**, qui, avec une syntaxe proche du Javascript et **6 verbes,** permet de définir de façon dynamique et déclarative les vues et leurs *bindings* (***Data***, ***Command*** et ***Layout***). Le développeur conçoit et développe son application avec ses deux DSL, du code standard .net ou Javascript et du HTML, CSS standard aussi.

Un petit exemple juste pour illustrer la logique du binding déclaratif

On va créer une application SPA avec 3 vues. La première vue **DemoProgrammez** aura deux zones, l'une pour loger la vue **Multiplication** qui permettra de faire une multiplication en utilisant un service local en Javascript, et l'autre pour loger la vue **Addition** qui fera une addition en utilisant un service serveur en C#. **Le code du service client en Javascript qui s'appelle** Calcul Fig.2. **Le code du service serveur en** C# qui s'appelle Calcul aussi Fig.3. **La définition de deux contrôles en** HTML5 Principal et Calcul **avec une propriété** Result Fig.4.

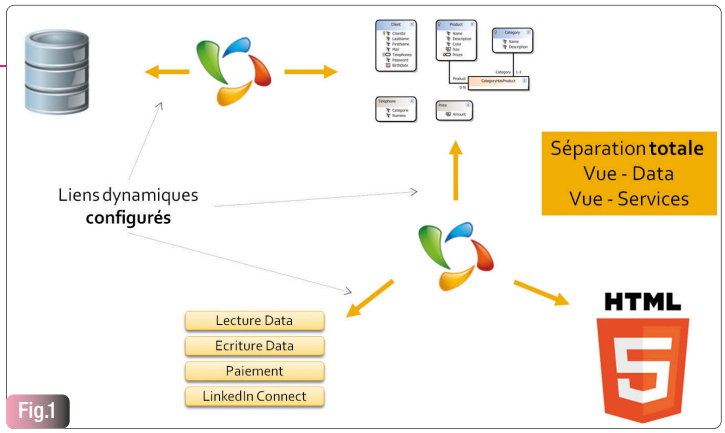


Fig.1

La création des vues et leurs binding avec le DSL

Ceci n'est pas du code. Ce sont des déclarations de ***binding***, Fig.5 présentées avec une syntaxe proche du Javascript pour pouvoir profiter de *l'intellisense* de Visual Studio. Comme on peut voir ici : Fig.6. Ou là : Fig.7. Le DSL met au bout des doigts du développeur toutes les informations dont il a besoin pour lier de façon déclarative les éléments les uns aux autres, Data, Contrôles, Vues, Services...

Enfin le point d'entrée du programme

Et le résultat une appli SPA Full Ajax sans code généré qui tourne dans le PaaS Azure : Fig.9.

Conclusion

On gagne du temps et **on diminue le volume de code à écrire** par un facteur important **- entre 20 et 50 fois -** en utilisant un moteur technique pour s'occuper de tout ce qui est prévisible et répétitif : la gestion des exceptions, des traces, des logs, la manipulation du DOM HTML, les conversions de données d'un format à un autre, les validations, le mode déconnecté, les échanges Ajax, les lectures/écritures des données physiques en SQL et en NoSQL, la gestion multilingue, l'intégration avec Azure, la gestion des versions et bien plus encore. Au delà du temps gagné, écrire moins de code c'est **écrire moins de bugs**, on gagne en qualité et on est beaucoup plus agile ; la maintenance est facile et la tolérance aux changements bien meilleure. C'est aussi une baisse des besoins en expertise technique, et plus de temps à consacrer à la compréhension du besoin et à l'expérience utilisateur. En utilisant la méta-programmation et l'AOP, l'approche Aspectize permet de développer avec les langages standards (HTML5, CSS, Javascript, .Net), et de produire des **applications métiers** spécifiques, modernes, sécurisées, performantes, consommables sur tout type de device, déployées en un clic, avec un processus ultra simple, itératif, et extrêmement agile.

```
/// <reference path="S:\Delivery\Aspectize.core\AspectizeIntellisense.js" />

Global.Calcul = {

    aasService:'Calcul',
    aasPublished:true,

    Multiplier: function (a, b) {

        return a * b;
    }
};
```
Fig.2

```
using Aspectize.Core;

namespace ExemplePourProgrammez {
    public interface ICalcul {
        int Ajouter (int a, int b);
    }

    [Service(Name = "Calcul")]
    public class Calcul : ICalcul {
        int ICalcul.Ajouter (int a, int b) {

            return a + b;
        }
    }
}
```
Fig.3

```
<div aas:control="Principal">

    <h1>Démo de binding distribué et dynamiques</h1>

    <div aas:name="Addition" aas:type="Panel"></div>
    <div aas:name="Multiplication" aas:type="Panel"></div>

</div>

<div aas:control="Calcul">

    <input name="A" type="text" /> <input name="B" type="text" />
    <input name="Op" type="button" /> = <span>{Result}</span>

</div>
```
Fig.4

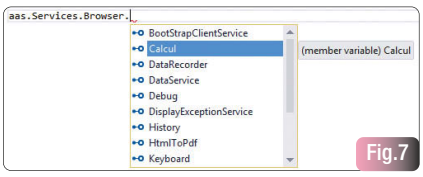


Fig.7

```
var demoCalcul = Aspectize.CreateView("DemoProgrammez", aas.Controls.Principal);

var vAddition = Aspectize.CreateView("Addition", aas.Controls.Calcul, aas.Zones.DemoProgrammez.Addition, true);

vAddition.Result.BindData("");
vAddition.Op.value.BindData("+");
vAddition.Op.click.BindCommand(aas.Services.Server.Calcul.Ajouter(vAddition.A.value, vAddition.B.value), aas.View.Addition.Result);

var vMult = Aspectize.CreateView("Multiplication", aas.Controls.Calcul, aas.Zones.DemoProgrammez.Multiplication, true);

vMult.Result.BindData("");
vMult.Op.value.BindData("*");
vMult.Op.click.BindCommand(aas.Services.Browser.Calcul.Multiplier(vMult.A.value, vMult.B.value), aas.View.Multiplication.Result);
```
Fig.5

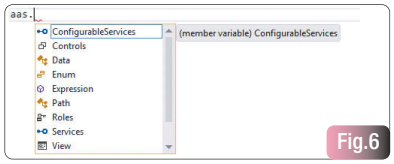


Fig.6

```
function Main() {

    Aspectize.Host.InitApplication();

    Aspectize.Host.ActivateViewByName('DemoProgrammez');
}
```
Fig.8

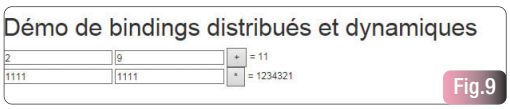


Fig.9

TypeScript : le langage en pleine ascension

Pour pouvoir parler de TypeScript, il est nécessaire de faire un point rapide à propos de JavaScript.

Etienne Margraff
Evangeliste web Microsoft

JavaScript est un langage basé sur une spécification : ECMAScript (ES). Chaque version de JavaScript est en fait une nouvelle implémentation de cette spécification. Pour vous donner une idée de la fréquence de mise à jour, ES3 est sorti en 1999. ES4 a été rapidement abandonné pour laisser la place à ES5 (une version plus simple et moins ambitieuse) en 2009. C'est la version de JavaScript utilisée à l'heure actuelle par la majorité des sites Web. ES6 a pour objectif d'ajouter les fonctionnalités initialement prévues dans ES4 et abandonnées pour ES5.

Bref, tout ça pour dire que ES6, c'est le futur de JavaScript (JS). Et ce futur est plutôt prometteur : le support du mot clé **class,** les **arrow function,** et tout un tas d'autres fonctionnalités vont simplifier l'écriture et la compréhension du code JS. Même si cette version de JS commence à être correctement implémentée par la majorité des navigateurs, ce n'était pas le cas il y a quelques mois. Et dans tous les cas, certains anciens navigateurs sont toujours dans la nature et ne supportent pas cette nouvelle version du langage.

Comment profiter d'une syntaxe similaire, tout en étant compatible ? C'est le rôle de **TypeScript** (www.typescriptlang.org).

TypeScript est un langage **Open Source** créé par Microsoft et plus particulièrement par Anders Helsjberg qui est également le créateur de l'ancêtre de Turbo Pascal, de Delphi et de C#. Le langage est basé sur les spécifications **ECMAScript 6,** entre autres, et permet d'utiliser les concepts de classes, d'héritage, d'accesseurs publics et privés, etc.

Attention cependant ! Ce langage n'a en aucun cas vocation à remplacer JavaScript. Il faut plutôt le voir comme une aide à l'écriture de JS propre qui utilise les patterns réputés pour être les meilleurs. On écrit le code avec **TypeScript** et il est ensuite compilé / traduit en JavaScript. On peut choisir la version cible d'ECMAScript en fonction de son besoin. ES5 pour le plus grand nombre des cas et ES6 pour faire des tests, ou si on développe pour une population d'utilisateurs qu'on sait utiliser un navigateur très récent.

L'apport du typage fort et de la structuration sous forme de classes est un atout considérable quand on travaille à plusieurs sur un projet. C'est intéressant pour des projets en entreprise, car c'est un milieu dans lequel on est habitué à ces principes. L'analyse du compilateur évite certaines erreurs qui sont si vite commises en JavaScript. Apprendre TypeScript est une tâche très simple, le site du langage contient un handbook avec toutes les explications pour chaque fonctionnalité. Comme pour la majorité des langages et frameworks Web, il y a également un playground qui permet de jouer en live avec TypeScript. Dans la partie gauche vous écrivez du code TS, et à droite, vous avez la version générée en JavaScript : plutôt pratique non ? Fig.1. Vous pouvez démarrer dans avoir besoin de configurer un environnement complet !

Fig.1

Les bases de TypeScript

TypeScript fonctionne de manière assez simple. Vos fichiers ne sont pas suffixés par **.js** mais par **.ts**. La phase de compilation dont on parlera un peu plus tard permet de générer le fichier .js qui portera le même nom.

Un des patterns les plus importants en JavaScript concerne la gestion des modules et l'isolation de code. En TS, il suffit d'utiliser le mot clé **module**. Chaque élément à l'intérieur de ce module doit être exporté pour être accessible en le préfixant du nom du module Fig.2.

```
Fig.2
module MonModule {
    export class MaClasse {
        greeting: string;
        constructor(message: string) {
            this.greeting = message;
        }
        greet() {
            return "Hello, " + this.greeting;
        }
    }
}
var greeter = new MonModule.MaClasse("world");
```

Les classes sont utilisées de manière assez proche de ce que l'on connait dans d'autres langages. On utilise le mot clé **class** pour la créer et on peut ensuite la composer avec différentes méthodes. Le constructeur est défini via le mot clé **constructor**.

```
var maVariable: number;

maVariable = 2;
maVariable = "";
```
Fig.3

TypeScript propose également la possibilité d'utiliser les modificateurs d'accès tels que **public** ou **private.** Cela aura un impact lors de l'utilisation d'environnement de développement exploitant l'auto-complétion pour TypeScript (les membres privés ne seront plus accessibles). Attention cependant, ceci n'est qu'une vue de l'esprit car ce n'est pas possible tel quel en JavaScript.

Le typage est possible sur le même principe que la majorité des langages typés. Vous pouvez utiliser un des types de bases comme **number** (pas de différence entre flottant, entier ou autres en TS), **string** ou **any** (équivalent de **object** en C#) par exemple, ou alors utiliser les types que vous avez créés à travers le mot-clé **class**. Le typage se fait après la déclaration de la variable, sur la même ligne : Fig.3.

Encore une fois ici, le typage ne sera utilisé que par le compilateur TypeScript pour vous éviter une erreur, comme vous pouvez le voir sur l'exemple.

Si vous assignez une valeur d'un type différent de celui prévu lors de la déclaration, le compilateur remontera une erreur.

Le langage ne s'arrête pas là et apporte également le support de l'héritage avec le mot clé **extends**, la gestion des **interfaces** et les génériques.

Fichiers de définition : quand vous utilisez une librairie externe qui vous est livrée en JavaScript et que vous voulez pouvoir l'utiliser dans du code TypeScript, il vous faut les définitions des types qui y sont. Pour cela, il

est nécessaire de télécharger un fichier .d.ts et de le mettre à disposition du compilateur. Vous trouverez ces fichiers sur github : https://github.com/DefinitelyTyped. Il y en a pour à peu près tout (jQuery, AngularJS, etc.).
Au final pour utiliser votre fichier, vous utiliserez le fichier .js classique généré par le compilateur, dans vos fichiers HTML.
Vous pouvez retrouver tout ça sur le guide : http://www.typescriptlang.org/Handbook
Le langage est déjà très avancé à l'heure actuelle, mais il est toujours en phase de développement. Comme c'est un projet Open Source, vous avez évidemment accès au code sur GitHub mais vous pouvez y voir le change log de la version actuelle (1.4) et voir notamment qu'elle a apporté le let et le const qui sont des mots clés introduits par ECMAScript 6. L'équipe fournit un énorme effort pour la version 1.5 qui contient de nombreuses nouveautés qui permettront d'atteindre un niveau de compatibilité avec ES6 plus poussé qu'aujourd'hui. On y retrouve notamment l'opérateur spread, les modules ES6, les symboles et la déstructuration qui permet d'assigner un ensemble de valeurs à un ensemble de variables d'un seul coup.
Voici par exemple une utilisation du spead operator :

```
var tableau: Array<number>;
tableau = new Array<number>();

var values = [1, 2, 3];

tableau.push(...values);
```

Cela aura pour effet de créer un tableau vide et d'appeler la fonction push sur la variable tableau en ***éclatant*** les valeurs du tableau.
Cela correspond au code suivant :

```
for(var i = 0 ; i < values.length ; i++)
{
    tableau.push(values[i]);
}
```

Vous l'aurez compris, l'objectif est de simplifier le travail du développeur. La version 1.6 va un peu plus loin car l'objectif de TypeScript n'est pas que de coller parfaitement à la spécification ES6. Par exemple, cette version inclura la gestion de l'asynchronisme basé sur les mots clés **async** et **await**. Il s'agira d'une implémentation du pattern **Promise** qui est habituellement écrit sur ce modèle :

```
maFonctionAsync().
        then(
    function() {
        //code à exécuter après le retour de la fonction maFonctionAsync.
    }
);
```

Cela évite de gérer trop de fonctions de callback comme on peut en avoir l'habitude en JavaScript.
L'utilisation des mots clés async et await permettra d'avoir une syntaxe encore plus simple :

```
await maFonctionAsync() ;
//l'exécution du code est stoppée tant que maFonctionAsync n'a pas terminé son traitement
```

Evidemment, cette syntaxe sera traduite en Javascript lors de la phase de compilation de TypeScript.
Cela augure beaucoup de choses intéressantes pour le futur de ce langage dont l'objectif est d'être en avance de phase sur ce qui est possible en JavaScript, tout en garantissant une compatibilité avec le plus grand nombre de navigateurs.

Industrialiser TypeScript

Même si ce n'est pas une obligation, un des moyens les plus agréables d'écrire en TypeScript est d'utiliser Visual Studio. Il s'agit d'un environnement de développement proposé par Microsoft. Dans le cadre de TypeScript, il propose l'autocomplétion, la compilation, l'exécution automatique de votre fichier html associé. Contrairement à ce que l'on pense souvent, il existe une version gratuite pour la communauté (Visual Studio Community Edition) qui permet d'accéder à ces fonctionnalités.
Si vous ne voulez pas profiter de Visual Studio ou que vous préférez utiliser un autre environnement de développement comme SublimeText ou Notepad++, vous pouvez installer les outils en lignes de commande pour pouvoir compiler facilement de TypeScript vers JavaScript. Les instructions sont disponibles sur le site www.typescriptlang.org mais vous pouvez tout simplement utiliser NodeJS Package Manager (npm) en exécutant la ligne de commande :

```
npm install –g typescript
```

Cela installe l'outil **tsc.exe** qui compile les fichiers .ts et génère le .js associé.

```
tsc monfichier.ts
```

Pour une industrialisation complète, les développeurs Web utilisent généralement des exécuteurs de tâches comme **grunt** ou **gulp**. Il est tout à fait possible d'intégrer la compilation TypeScript dans une tâche de l'un ou l'autre de ces outils via **grunt-typescript** ou **gulp-typescript**.

Et le debug ?

C'est une question légitime et importante quand on aborde un nouveau langage. Visual Studio permet de s'attacher à un navigateur et de faire du debug pas à pas dans le code TypeScript. C'est également le cas dans les outils de debug d'Internet Explorer ou Chrome (via F12). Cela fonctionne grâce à un fichier .map qui est généré lors de la compilation et permet de faire le lien entre le code JavaScript et le code TypeScript correspondant. Le debuger sait alors associer un morceau de .ts à un morceau de .js pour réaliser le pas à pas et les points d'arrêt dans le code.

TypeScript dans Angular.JS

Depuis quelques temps, Google travaille avec Microsoft pour pouvoir utiliser TypeScript pour développer Angular.JS. Il y a plusieurs raisons à cela, et notamment le fait que cela permet de travailler en équipe plus facilement. A l'origine, Google a créé sa propre implémentation de TypeScript : atScript. Depuis novembre 2014, c'est TypeScript qui remplace officiellement ce langage et qui devient la référence pour http://angular2.com/ . Ceci est une marque de confiance importante apportée au langage et nous conforte en tant que développeur dans sa pérennité.
C'est également pour Angular un bon moyen de proposer une syntaxe équivalente à ECMAScript 6 tout en continuant à fonctionner sur d'anciens navigateurs en compilant le code vers ES5.
Si vous voulez utiliser TypeScript pour développer une application avec Angular, ça sera également facilité car Google vous fournit tous les fichiers .d.ts dont vous avez besoin. Ceci ne sera évidemment pas une obligation, mais pourquoi se priver ?
Si voulez en savoir plus sur la raison du passage à TypeScript, vous pouvez regarder cette vidéo de la conférence Angular (ng-conf) : http://bit.ly/1FFGRoH

Et Dart alors ?

Mathieu Lorber
Freelance / CTO Instill.io

À Instill.io, nous sommes partis il y a un peu plus d'un an à la recherche de la perle rare pour construire notre éditeur de vidéo informative en "single-page application", en JavaScript dans le navigateur. Nous avions de l'expérience avec différentes technologies, mais le changement c'était maintenant. Cet univers est en renouvellement constant depuis plusieurs années : l'explosion des frameworks et "transpilers" JavaScript est continu, l'ouragan Angular n'y change pas grand chose. Cet environnement reste aussi riche qu'instable, il est objectivement aujourd'hui impossible de trouver une technologie qui réunisse les trois qualités d'être mature, pérenne et productive.

Une initiative nous a plu par son audace et son approche. Habitués à des applications assez grosses, nous sommes convaincus que JavaScript pose rapidement des problèmes de maintenabilité. Sans compter l'outillage assez rebutant. Nous nous sommes ainsi intéressés à Dart, nouveau langage poussé par Google, avec pour ambition ultime de se positionner comme une alternative native à Javascript. L'objectif : un autre langage, branché via les mêmes API au navigateur, bref, rien à voir avec une applet.

Mais comment l'amener là ? Simple, il est désigné dès le départ pour compiler en JS, et donc fonctionne sur les navigateurs actuels. Cette approche nous a plu. JavaScript n'évoluera quoi qu'il arrive pas sans compilation !

Dart est un langage moderne, typé optionnellement, qu'un programmeur C# ou Java apprendra très rapidement. La syntaxe reste proche de JS, en version encore plus succincte. Les fonctionnalités du langage sont supérieures à celles des prochaines versions de Javascript, non disponibles avant plusieurs années. Les API du navigateur, accusant le poids de l'âge, sont rendues plus cohérentes. Système de build, packaging et gestion des dépendances en natif.
Le SDK est livré avec un Chrome qui intègre la VM native. Si cette intégration restera finalement réservée au SDK, le confort au développement est réel : la VM Dart est plus rapide que les moteurs Javascript actuels.

Aujourd'hui, le compilateur est mature et produit un résultat au poids raisonnable - pourvu qu'on respecte certaines règles. L'environnement de travail est agréable à utiliser : l'éditeur fourni avec le SDK est un Eclipse allégé qui fait son boulot. Pour encore mieux, il faut aller voir le plugin WebStorm/Intellij.

Si Dart est un langage à l'avenir incertain - la concurrence est rude et les velléités politiques extrêmement fortes - il nous offre aujourd'hui un environnement de travail sans comparaison. L'investissement de Google est tel que le worst case scenario est une conversion sur la solution qui remportera la bataille de ces prochaines années. Bref, Dart était un choix osé, mais nous n'avons pas à le regretter aujourd'hui !

L'avenir de C# 7 : la communauté participe et décide avec Microsoft

Depuis les dernières annonces majeures faites par Microsoft de publication de toutes les sources du framework .NET sur Github, la firme de Redmond montre jour après jour son désir de travailler avec la communauté. Il est, par exemple, possible de proposer des modifications du code et d'interagir directement avec les équipes d'architecture et de développement du framework. Le langage C# et son design sont évidemment aussi concernés. Cet article va vous éclairer sur cette nouvelle façon de travailler et comment vous pouvez y participer.

Jason De Oliveira
CTO | MVP C# chez **Cellenza**
Cellenza - Software Development Done Right
Son Blog: http://www.jasondeoliveira.com

Fathi Bellahcene
Manager | MVP C# chez **Cellenza**
Cellenza - Software Development Done Right
Son Blog: http://blogs.codes-sources.com/fathi

C# 6 à peine dévoilé, Microsoft s'attèle déjà à définir les contours de la prochaine version du langage, mais cette fois-ci de manière publique et totalement transparente ! Voici donc les nouvelles fonctionnalités et directions de C# 7 ainsi que les pistes qui sont étudiées actuellement.

Organisation et processus de validation des nouvelles fonctionnalités

Microsoft a défini un nouveau processus de qualification, de spécification et de validation des nouvelles fonctionnalités pour la prochaine version de C#. Ce processus est axé sur la collaboration avec la communauté. Il est alors possible de participer à chaque étape et de rajouter ses propres idées après validation par Microsoft.

Une Design Team pour mener les discussions

Pour cette tâche difficile, Mads Torgersen (Language Product Manager) a formé une équipe d'une dizaine de personnes qui se réunit toutes les semaines pour discuter et valider les nouvelles fonctionnalités. Cette équipe compte, bien évidemment, Anders Hejlsberg (créateur du langage C# et Chief Language Architect) qui a la lourde charge de trancher en cas de désaccord.

Transparence et ouverture à la communauté

Tous les comptes rendus des réunions sont publiés sur Github et peuvent donc être commentés et annotés par la communauté. De la même manière, il est possible de proposer des idées via Github et de voter pour celles que vous trouvez intéressantes.
Lorsqu'une nouvelle fonctionnalité est jugée pertinente par l'équipe de design, un « Owner » est désigné et la spécifie de manière transparente et ouverte sur Github. Il faut alors l'analyser et la tester dans tous les contextes possibles.

Un processus démocratique, mais Microsoft garde le dernier mot

Il est important de préciser qu'à ce stade, la nouvelle fonctionnalité n'a encore aucune garantie d'être intégrée au framework. A la fin de la phase de spécification, la Design Team prend la décision finale de l'intégrer (ou pas) sans forcément prendre en compte les votes et l'avis de la communauté.
Cette organisation est plutôt avantageuse car elle permet de définir une ligne cohérente. Si l'on prend comme exemple l'évolution des langages comme HTML ou Java, on remarque qu'ils évoluent lentement, enlisés dans des discussions interminables.
Toutefois, on peut rester confiants sur le fait d'être entendu (comme ce fut le cas sur bien des points dans le passé) ainsi que sur la qualité des décisions qui seront prises.
Voici le lien de la discussion sur Github :
https://github.com/dotnet/roslyn/issues/assigned/MadsTorgersen?page=3&q=is%3Aopen+is%3Aissue+assignee%3AMadsTorgersen

Les thèmes des nouvelles fonctionnalités

Microsoft a défini un certain nombre de thèmes principaux. Les propositions de nouvelles fonctionnalités s'y retrouvent avec leurs priorités et leur état d'avancement. Ci-dessous les thèmes qui existent actuellement et leurs propositions principales.

Travailler avec des données

Les applications d'aujourd'hui traitent souvent d'énormes quantités de données très structurées. La version actuelle de C# n'est pas vraiment très bien adaptée pour cela car elle est plutôt centrée sur le comportement (« behavior-driven ») au lieu d'être centrée sur les données (« data-driven »). Les langages fonctionnels comme le F# peuvent alors servir comme bon exemple.
Voici quelques propositions des nouvelles fonctionnalités:

- Pattern Matching
- Tuples
- Extension Members
- Slicing
- Immutabilité

Performance, fiabilité et interopérabilité

Le langage C# a déjà beaucoup évolué depuis ses premières versions au niveau des performances, de la fiabilité et de l'interopérabilité, mais il y a encore plusieurs pistes d'amélioration pour rendre tout cela encore plus efficace.

Componentization du framework

Ce thème est plutôt orienté outils et vise à redéfinir comment factoriser et combiner des applications basées sur le .NET framework.
Voici quelques propositions des nouvelles fonctionnalités:

- Reference Assemblies
- Static linking à la place de IL merge

- Déterminisme
- Support pour NuGet
- Versioning et adaptive light-up

Distribution

Ce thème regroupe les sujets et fonctionnalités spécifiques concernant le calcul distribué.
Voici quelques propositions des nouvelles fonctionnalités :

- Sequences Async
- Serialization

Méta-programmation

La méta-programmation est un sujet important depuis longtemps déjà. La sortie de Roslyn a vulgarisé son utilisation en la rendent facile, accessible et efficace. L'implémentation au niveau du langage C# n'est toutefois pas optimale et doit évoluer à l'avenir.
Voici quelques propositions des nouvelles fonctionnalités:

- Virtual extension methods
- Generic constructor constraints
- Delegate et enum constraints
- Operators ou object shapes comme constraints

Null

Valider que les objets ne soient pas « null » pour assurer le bon fonctionnement des applications est une tâche répétitive et fastidieuse qui n'apporte pas de vraie valeur. Il y a plusieurs propositions pour faciliter et automatiser cette tâche et ainsi rendre les développeurs encore plus productifs.
Voici quelques propositions des nouvelles fonctionnalités:

- Non-nullable reference types
- Safe nullable reference types

Themeless in Seattle

Ceci est le thème qui regroupe les fonctionnalités qui ne peuvent pas facilement être classées dans les autres thèmes.
Voici quelques propositions des nouvelles fonctionnalités:

- Type providers
- Scripting
- IEnumerable params
- Binary literals et digit separators

Cette liste de thèmes va, bien entendu, évoluer et, comme vous l'avez vu, de nouvelles fonctionnalités qui ne peuvent pas être facilement classées, mais qui font sens pour une future version de C#, vont quand même être spécifiées, validées et implémentées.

Les nouvelles fonctionnalités actuellement en discussion

Voici une liste non exhaustive de nouvelles fonctionnalités en phase de spécification qui semblent très intéressantes.

Pattern Matching

Le Pattern Matching est un mécanisme bien connu et très apprécié des développeurs F#. Il permet de vérifier si un objet a un aspect particulier, et si oui, de l'extraire et de pouvoir le manipuler en tant que tel.
Il y a au moins deux cas utiles d'utilisation de ce pattern qui rendent le code plus lisible. Le premier est la validation lorsqu'un objet matche avec un pattern donné dans le cadre d'une instruction « if », comme dans les exemples d'implémentation suivants :

```
if (o is Point(*, 5) p) Console.WriteLine(p.x); //p doit être un point avec Y = 5
if (o is Point p) Console.WriteLine(p.x); //p est juste un point
if (p is (var x, 5)) ... //p est un type anonyme
```

Dans les exemples, le matching et l'assignation sont faits dans la même instruction. On utilise le mot clé « is » déjà existant en C#, qui a pour but de tester la correspondance avec un pattern. Il suffit juste de l'étendre.
Le second cas est encore plus utile dans le cadre d'une instruction « switch/case » :

```
switch (o) {
case string s:
  Console.WriteLine(s);
  break;
case int i:
  Console.WriteLine($"Number {i}");
  break;
case Point(int x, int y):
  Console.WriteLine("({x},{y})");
  break;
case null:
  Console.WriteLine("<null>);
  break
}
```

Cela implique de gros changements dans le comportement de l'instruction « switch/case », car il faut être en mesure de supporter tous les types dans la partie « switch », et les patterns dans la partie avec les « case ». Il y a une question qu'on pourrait se poser : comment, dans une classe, maîtriser le matching avec d'autres types, et comment construire un objet à partir de ces types ? Une proposition d'implémentation serait d'avoir des méthodes statiques nommées « Match(...) » qui se chargeraient de renvoyer le résultat du matching vrai/faux ainsi que l'objet résultant. Voici un exemple :

```
class Point {
  public Point(int x, int y) {...}
  void Deconstruct(out int x, out int y) { ... }
  static bool Match(Point p, out int x, out int y) ...
  static bool Match(JObject json, out int x, out int y) ...
}
```

Voici le lien de la discussion sur Github :
https://github.com/dotnet/roslyn/issues/1572

Types non nullables

En général, les développeurs passent beaucoup de temps à valider que les objets qu'ils utilisent ne soient pas « null » pour assurer le bon fonctionnement de leurs applications.

```
if (o != null) Throw new NullReferenceException(...);
```

Pour améliorer cela, une proposition consiste à utiliser le symbole « ! » pour indiquer qu'un type ne peut pas être « null ». Voici un exemple pour la classe Point :
Point p : référence classique, rien ne change.
Point! p : p (référence obligatoire) ne peut en aucun cas être nul.
Point? p : p est une référence nullable.
Cela signifie que lorsque vous créez un objet obligatoire, le compilateur

va vérifier que cet objet ne peut pas être « null ». S'il est instancié, à partir d'une méthode par exemple, le compilateur va s'assurer qu'elle renvoie obligatoirement un objet non « null ». Dans le cas inverse, il y aura une erreur à la compilation. Ainsi, le code suivant produit les erreurs associées :

```
var! dog = new Dog("Sam"); // var est Dog!
var! dog1 = MethodReturningMandatoryRef(); // var est Dog!
var! dog2 = MethodReturningNullableRef(); // Compiler Error
var! dog3 = MethodReturningGeneralRef(); // Compiler Error
```

Il est évidemment prévu l'utilisation de ce symbole avec le mot clé « var » ainsi que les génériques, mais cela risque de provoquer parfois de la confusion chez les développeurs.
Voici le lien de la discussion sur Github :
https://github.com/dotnet/roslyn/issues/227

Type Immutable

Actuellement, il existe de nombreuses manières de forcer un objet à être immutable. Cela signifie qu'une fois qu'il a été construit, il n'est pas possible de modifier ses propriétés. On peut le faire en utilisant le mot clé « readonly » ou encore en spécifiant des propriétés qui ont des méthodes « set » privées comme ci-dessous :

```
public class Person
{
  public Person(string firstName, string lastName, DateTimeOffset birthDay)
  {
    FirstName = firstName;
    LastName = lastName;
    BirthDay = birthDay;
  }

  public string FirstName { get; }
  public string LastName { get; }
  public DateTime BirthDay { get; }

  public string FullName => $"{FirstName} {LastName}";
  public TimeSpan Age => DateTime.UtcNow – BirthDay;
}
```

Ici, un développeur n'a aucun moyen d'expliciter son intention de rendre cette classe immutable. Un autre développeur ne pourra donc pas savoir qu'il faut la traiter comme telle, et une modification anodine peut avoir des conséquences très négatives dans un contexte « multi-thread ».
Dans l'exemple, un développeur souhaite ajouter une méthode « set » publique à cette classe comme on peut le voir ci-dessous. Il risque alors d'avoir des erreurs au runtime... ce qu'il faut impérativement éviter !

```
public class Person
{
  public Person(string firstName, string lastName, DateTimeOffset birthDay)
  {
    FirstName = firstName;
    LastName = lastName;
    BirthDay = birthDay;
  }

  public string FirstName { get; }
  public string LastName { get; }
  public DateTime BirthDay { get; set; } // rend la classe mutable!

  public string FullName => $"{FirstName} {LastName}";
  public TimeSpan Age => DateTime.UtcNow – BirthDay;
}
```

Un exemple de modification de cette classe aux conséquences similaires mais un peu plus sournois serait d'ajouter des références à la classe elle-même :

```
public class Person
{
//…
  public Person[] Ancestors { get; }; // rend la classe mutable!
//…
}
```

De la même manière, vous pouvez trouver d'autres exemples de modifications qui peuvent sembler sans risque mais qui, dans des contextes spéciaux, peuvent être néfastes (comme en programmation parallèle). C'est pourquoi, la fonctionnalité permettant explicitement d'indiquer qu'une classe est immutable semble justifiée et souhaitable.
La solution pourrait-être extrêmement simple car l'ajout du mot clé « immutable » devant la classe (ou d'une structure) permettrait d'assurer à la compilation la vérification du caractère réellement immutable du type.

```
public immutable class Person
{
//…
}
```

Le compilateur va :

- Implicitement marquer tous les champs (fields) comme « readonly ».
- S'assurer que tous les champs sont eux-mêmes des types immutables (comme par exemple les types de base du framework suivant : Int32, Double, TimeSpan, String,...).
- Vérifier que l'utilisation du mot clé « this » est limitée dans le constructeur à des instructions de lecture/écriture des champs.

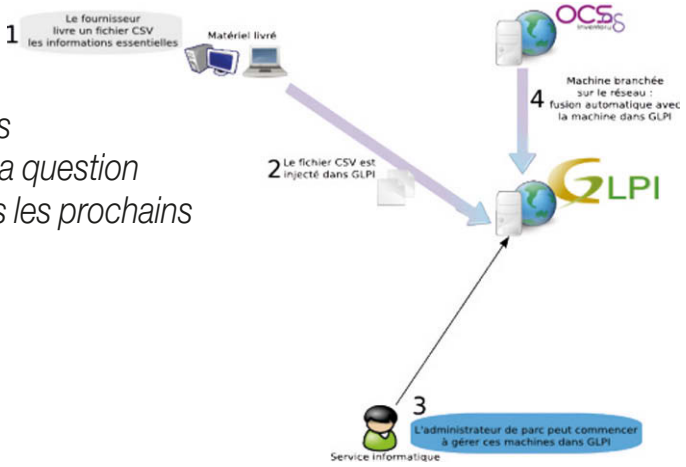
Voici le lien de la discussion sur Github :
https://github.com/dotnet/roslyn/issues/159

Conclusion

Vous avez maintenant vu le nouveau processus de qualification, de spécification et de validation des nouvelles fonctionnalités pour la prochaine version de C#. Tout est tourné vers la collaboration avec la communauté. Vous aurez également noté que beaucoup de ces nouveautés proviennent des langages fonctionnels et que C# va se rapprocher de plus en plus de F#. Certaines évolutions déjà proposées sont très intéressantes et vont grandement nous faciliter la vie.
N'hésitez pas à donner votre avis et à vous exprimer sur ces nouvelles fonctionnalités en votant sur le Github (https://github.com/dotnet/roslyn). Vous pouvez même proposer les fonctionnalités qui vous manquent cruellement dans votre quotidien en espérant qu'elles soient retenues pour une future version de C#.

En route vers PHP 7

Quand un langage Web est en tête des hit-parades depuis de nombreuses années, on peut se poser la question si celui-ci restera toujours dans cette position dans les prochains mois, voire années.



Christophe Villeneuve
consultant IT pour Neuros, auteur du livre "Drupal7 en mode avancé" aux éditions Eyrolles et auteur Editions ENI, Rédacteur pour WebRIVER, membre des Teams DrupalFR, AFUP, LeMug.fr, Drupagora, PHPTV..

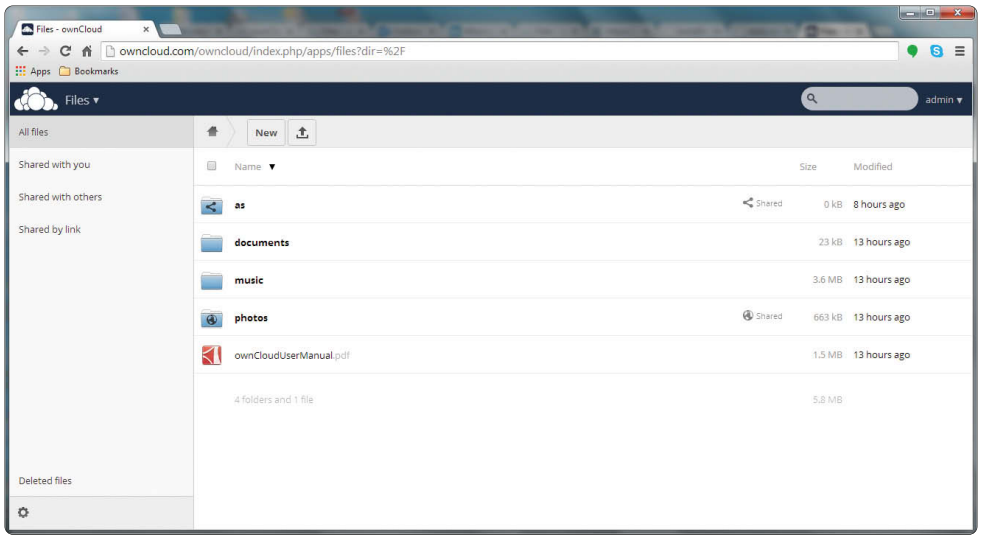
Depuis son origine, le langage PHP a toujours voulu rester un langage d'apprentissage facile, au détriment de certains autres langages. Cependant, la facilité d'utilisation permet d'offrir aux internautes des offres partagées et mutualisées, ou encore de monter son environnement chez soi. Si vous suivez ce langage, la courbe a toujours été orientée à la hausse comme la montre W3Techs (http://www.w3techs.com). Bien entendu, certains secteurs sont plus à même d'utiliser ce langage pour répondre à certaines demandes bien précises par rapport à d'autres.

La toile d'araignée PHP

L'Internet est souvent représenté par une toile d'araignée, et le langage PHP est un des maillons de cette toile. Celui-ci sait se greffer sur les nouvelles tendances pour passer inaperçu dans vos usages de tous les jours. Grâce à sa légèreté et sa robustesse, le langage répond aux attentes des multi-utilisateurs et des multi-comptes complexes. Cela se traduit par des milliers d'applications, dont voici quelques secteurs qui l'utilisent.

Le hardware

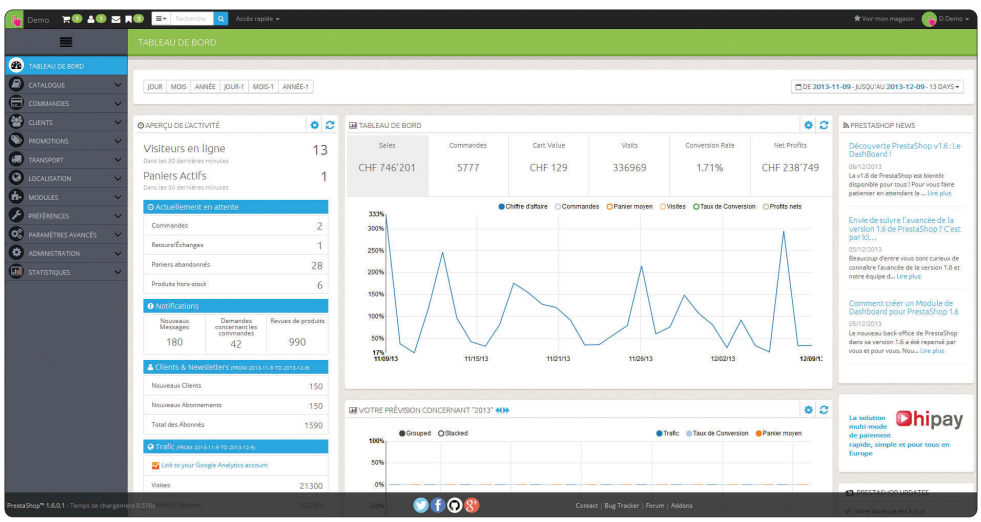
Le support magnétique est un des composants de la micro-informatique, et si vous souhaitez stocker vos fichiers, les diffuser et les partager, vous le faisiez à l'époque sur disquettes, CD, disques magnétiques... Même si l'opération reste toujours identique, le Web a permis de faciliter la notion de partage et de diffusion, grâce à l'hébergement sous la forme d'espaces dédiés ou mutualisés. Le langage PHP a su se positionner dans cet environnement grand public comme une brique pour répondre à une infrastructure mutualisée ou dédiée. Comme ceci, la gestion des multi-comptes et des multi-espaces dans un même espace de stockage a été facilitée et rendue possible.



Cependant, de nos jours, la technologie s'est déportée vers le Cloud pour proposer des services déportés où le matériel hardware n'est plus la raison première du fonctionnement classique du Web au niveau du stockage, mais s'est plutôt tournée vers l'applicatif.

Pour répondre à ces nouvelles demandes, le langage PHP propose de nombreuses applications, comme OwnCloud (www.owncloud.org) qui est un logiciel libre offrant une plateforme de services de stockage et partage de fichiers en ligne.

Par ailleurs, pour gérer le stockage ou plusieurs stockages, vous prévoyez la surveillance d'un réseau. Pour cela, il existe GPLI (www.glpi-project.org), qui est une application pour gérer un parc informatique et de servicedesk.



Framework spécialisé

Les frameworks spécialisés sont l'association de fonctionnalités PHP avec un but précis comme la gestion de contenu, de blogs, de sites marchands (B2B, B2C)... Grâce à eux et au langage PHP, ils ont permis de simplifier l'utilisation du Web et par conséquent ont contribué à son succès.

La gestion de contenu

La gestion de contenu (CMS) permet de répondre à une attente des internautes car ils s'occupent principalement des contenus des sites Webs. Ainsi, vous avez à disposition des applications comme Drupal (www.drupal.org), Wordpress (www.wordpress.org), Joomla (www.joomla.org), Dotclear (www.dotclear.org).

Le B2B/B2C

La partie Business est aussi un facteur important avec la mise en place de portails marchands.
Ils se présentent sous différentes formes. Tout d'abord, l'ajout de plugins pour les CMS (voir paragraphe précédent). Ensuite, sous la forme d'applications spécifiques comme Prestashop (www.prestashop.com), Magento (www.magento.com).

ERP / CRM

La gestion administrative est un critère à ne pas négliger, car c'est une garantie du bon fonctionnement d'une entreprise, d'une association ou d'une utilisation personnelle.
Ce mode de gestion répond aussi bien à la problématique de management ou de celle de la gestion commerciale. Vous pouvez trouver Dolibarr (www.dolibarr.org), SugarCRM (www.sugarcrm.com)...

Autres secteurs

Comme beaucoup de webmasters ou de mainteneurs de sites Internet, vous êtes à la recherche d'informations pour obtenir des statistiques sur vos visiteurs, les pages les plus fréquentées, etc. Pour cela, il existe Piwik (www.piwik.org) qui mesure et analyse pour vous l'audience d'un ou plusieurs sites Webs.

Par ailleurs, il est parfois intéressant de posséder un forum, ce qui vous permet d'offrir une base de connaissances que vous partagez avec vos collègues ou les internautes pour rendre une partie interactive à votre projet. Pour cela, vous vous appuyez sur les forums comme PHPBB (www.phpbb.com) ou encore par l'intermédiaire de wiki, comme mediaWiki (www.mediawiki.org)

2015 le changement

L'année 2015 est une année importante pour le langage PHP :

- Tout d'abord, le 8 juin il fêtera ses 20 ans d'existence.
- Arrivée de la nouvelle version majeure : PHP 7

Ce changement ne va pas révolutionner le Web que nous connaissons et pratiquons tous les jours. Mais il va répondre à de nombreuses attentes des développeurs et développeuses qui contribuent à améliorer l'Internet de tous les jours. Pour cela, une nouvelle implémentation du coeur de PHP pour remplacer le Zend Engine actuel, appelée PHPNG (PHP Next Generation) est arrivée. Détails de PHPNG sur https://wiki.php.net/phpng
L'intérêt de ce changement est un gain de performances de l'ordre de 20 à 40 % sur des applications 'réelles' grand public, par exemple : Magento, Drupal, SugarCRM, Wordpress... Au niveau des conséquences suite à ce gain de performance, la consommation de la mémoire va diminuer, car elle sera moins gourmande.
Bien entendu, la contrepartie concerne les extensions PECL qui utilisent la mémoire pour améliorer leurs fonctionnalités et devront se mettre à jour pour pouvoir rester compatibles avec cette nouvelle version de PHP.
En plus de ce que vous connaissez déjà, la version 7 du langage reste toujours dans la même orientation pour le Web. Ainsi, de nouvelles fonctionnalités arrivent dans celui-ci, comme l'uniformisation des caractères.

Unicode Codepoint Escape Syntax

Il s'agit d'une fonctionnalité pour une utilisation de séquences Unicode dans les chaînes de caractères. L'utilisation de cette fonctionnalité passe par l'identification d'un code caractère encapsulé des caractères suivants : { et }

Il ne possède pas de limite au niveau du nombre.

```
<?php

echo "\u{202E}Reversed text"; // Affiche un texte de droite à gauche

echo "ma\u{00F1}ana"; // affiche le caractère suivant ñ

echo "man\u{0303}ana"; // "n" combiné avec le caractère ~ (U+0303)

?>
```

Bien entendu, de nombreuses autres fonctionnalités seront disponibles

Conclusion

Comme vous pouvez le voir, le langage PHP est avant tout un langage 100 % Web, ce qu'il fait très bien. Cependant, même si vous pouvez l'utiliser à d'autres fins applicatives, il laisse sa place aux autres technologies et langages, qui sont plus adaptés. Mais ce qu'il ne faut pas perdre de vue, c'est qu'il restera toujours présent tant que le Web vivra. Enfin, les applications présentées aujourd'hui sont référencées sur le site Framasoft avec beaucoup d'autres.

PHP 7 en 5 points

Proposed Milestones		
Milestone	**Timeline**	**Comment**
1. Line up any remaining RFCs that target PHP 7.0.	Now - Mar 15 (4+ additional months)	We're already well under way with doing that, with the PHPNG, AST, uniform variable syntax, etc.
2. Finalize implementation & testing of new features.	Mar 16 - Jun 15 (3 months)	
3. Release Candidate (RC) cycles	Jun 16 - Oct 15 (3 months)	Subject to quality!
4. GA/Release	Mid October 2015	Subject to quality!

1 : disponible dès octobre 2015. Les fonctionnalités sont désormais fixées
2 : introduction d'une nouveau opérateur

```
// Integers
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
```

3 : nouveauté dans les déclarations
4 : promesse de meilleures performances (25 à 70 % selon Zend)
5 : PHPNG : le nouveau cœur de PHP
PHP 7 sera aussi l'occasion de faire un peu de ménage comme par exemple de retirer les constructeurs PHP 4 qui seront obsolètes dans la v7 et totalement retirés dans la v8. On aura droit à JSOND à la place de JSON.
Les pré-versions de PHP7 sont disponibles sur :
https://github.com/rlerdorf/php7dev

La rédaction.

Programmation par Contrats : sûreté, sécurité, fiabilité

La problématique première lors du design d'un langage de programmation est traditionnellement l'expressivité en termes de description d'algorithmes et d'architecture. Rares sont ceux qui se soucient de la question de la spécification logicielle ou de la vérification. Et pourtant. On ne compte plus les coupures de presse annonçant pertes abyssales ou trous de sécurité béants laissés ouverts par des défauts de programmation souvent primaires. Le législateur logiciel contraint de plus en plus d'industries à des phases de tests et de relecture de code si étendues qu'elles accaparent le plus gros des ressources de développement. Dans ces domaines, toute l'aide que peut apporter le formalisme logiciel est la bienvenue. La clef de voute de la fiabilité que nous allons présenter ici, c'est la spécification. En particulier dans sa capacité à exprimer des propriétés sur le code sous forme d'assertions vérifiables : le contrat.

Quentin Ochem
Technical Account Manager / Ingénieur Commercial - AdaCore

Contrats sur types...

Le langage Ada fait ici figure d'exception, et en particulier sa dernière mouture Ada 2012. Il s'agit de l'un des rares langages à séparer de manière formelle une zone exclusivement dédiée à la spécification d'une unité. Cette zone peut contenir de nombreuse déclarations, des fonctions bien évidemment, mais également des types. Ces types peuvent être associés à des contraintes, en particulier une plage de valeurs. Dans l'exemple suivant (en C), on cherche à s'assurer que le domaine de variables relatives à la vitesse est toujours compris entre 0 et 200. On écrira alors typiquement du code défensif pour s'assurer de ne pas sortir des bornes :

```
#DEFINE MAX_SPEED = 200
#DEFINE MIN_SPEED = 0

void setAutoCruise (float speed) {
  if (speed > MAX_SPEED || speed < MIN_SPEED) {
    return;
  }

  // code
}

void setSpeed (float speed) {
  if (speed > MAX_SPEED || speed < MIN_SPEED) {
    return;
  }

  // code
}
```

On remarque que ce code est reproduit à plusieurs endroits, à priori à chaque fois que l'on manipule une valeur associée à la sémantique « vitesse ». En Ada, on préfèrera introduire une spécification de type, et utiliser ce type pour chaque donnée représentant la grandeur correspondante, en particulier les paramètres :

```
type Speed_Type is new Float range 0.0 .. 200.0;

procedure Set_Auto_Cruise (Speed : Speed_Type);
procedure Set_Speed (Speed : Speed_Type);
```

On parle ici de typage fort. La force de cette notation est que le compilateur peut générer des vérifications dynamiques de cohérence. Ainsi, un développeur peu soucieux pourra écrire :

```
procedure Accelerate_Auto_Cruise is
begin
  Set_Auto_Cruise (Current_Speed + 1.0);
end Accelerate_Auto_Cruise;
```

Ceci implémentant donc une fonction d'accélération naïve, qui augmente la vitesse d'un km/h sans tester de borne. Grâce au typage fort, le compilateur génèrera une vérification de cohérence, et lèvera une exception si la vitesse demandée est supérieure à 200. Charge à l'architecture logicielle d'être en mesure de répondre à ce type d'exception, peut-être en enregistrant l'erreur, et en déconnectant le régulateur de vitesse jusqu'à la prochaine mise à jour. On objectera peut-être que ce type de vérification est typiquement effectué par l'introduction de code défensif.
La force de cette approche, c'est que la défense est directement liée à la spécification, elle est visible, vérifiable, et son implémentation n'est pas soumise à la diligence du développeur.
Ada 2012 étend la spécification de contraintes de type via la notion de prédicat, permettant d'associer à un type une contrainte sous la forme d'une expression booléenne arbitraire. Par exemple :

```
type Stack is record
  Data : Data_Access;
  Size : Integer;
end record
with Dynamic_Predicate =>
  (if Data /= null then Stack.Size <= Stack.Data'Length
                   else Stack.Size = 0);
```

Le type ci-dessus est une structure de données modélisant une pile, contenant un pointeur sur un tableau de valeurs (Data) et le nombre d'éléments contenus dans la pile (Size). Le prédicat associé à ce type décrit une contrainte entre les deux champs.
Il spécifie entre autres que si le pointeur n'est pas nul, le nombre d'éléments contenu doit être inférieur à la taille du tableau. On imagine aisément le type d'erreur de mise à jour qui pourrait conduire à une incohérence à ce niveau.

... ou contrats sur fonctions

Mais le typage fort n'est pas tout. La forme la plus courante de contrats est en réalité portée par les fonctions (ou sous-programme en Ada). Il s'agit de vérifier des contraintes avant appel (la précondition) et de donner des garanties après (la postcondition). À ce titre, les contrats sur types permettent déjà d'exprimer des contraintes sur les paramètres. Les préconditions et postconditions permettent de les préciser, ou d'établir des relations entre paramètres, entre un état d'entrée et un état de sortie, voire même sur l'état du système.

Prenons un premier exemple simple. La fonction Get_Char décrite ci-dessous renvoie le caractère du tableau Line à l'index First_Char, puis incrémente First_Char. Le code en Ada donne :

```
function Get_Char return Character is
  C : Character;
begin
  C := Line (First_Char);
  First_Char := First_Char + 1;
  return C;
end Get_Char;
```

Passons sur la faute de goût manifeste concernant l'utilisation de variables globales au dépens de paramètres, c'est pour la science. Ada génère ici une vérification sur l'accès au tableau Line, qui lève une exception en cas de dépassement. Afin d'éviter une telle exception, un développeur sera tenté d'écrire un code de protection (ou défensif) :

```
function Get_Char return Character is
  C : Character;
begin
  if First_Char in Line'Range then
    C := Line (First_Char);
    First_Char := First_Char + 1;
    return C;
  else
    return ASCII.NUL;
  end if;
end Get_Char;
```

Ceci étant, bien qu'évitant un cas d'exception, ce code défensif introduit potentiellement une erreur dans le code. Considérons la boucle d'utilisation suivante :

```
loop
  Current_Char := Get_Char;
  exit when Current_Char in 'A' .. 'Z';
end loop;
```

Arrivée en fin de tableau, la boucle devient infinie. Le code défensif a ici introduit une erreur dans le code. Notons que la vraie erreur est d'appeler Get_Char sans considérer la condition de fin de tableau. Afin de nous assurer que le développeur le prend on compte, nous allons rajouter des contrats sur ce code. Une précondition, s'assurant que First_Char est un index du tableau, et une postcondition, mentionnant l'incrément de First_Char à chaque appel :

```
function Get_Char return Character
  with Pre => First_Char in Line'Range,
       Post => First_Char = First_Char'Old + 1;
```

La notation 'Old permet ici de référencer une valeur avant l'appel, et donc de comparer un état d'entrée et un état de sortie. Le lecteur capricieux objectera peut-être que la postcondition ne fait ici que paraphraser une portion du code. La simplicité de l'exemple est effectivement mise en défaut. A titre informatif, voici un exemple de post condition décrivant une contrainte sur une fonction de recherche, spécifiant que si un élément existe dans le tableau, il doit être retourné :

```
function Search (A : Integer_Array; Value : Integer) return Natural
with Post =>
  (if Search'Result = 0 then
    (for all Index in A'Range => A (Index) /= Value)
   else A (Search'Result) = Value));
```

En termes plus simples, si le résultat est 0, tous les index du tableau A sont de la forme « l'élément de A à l'index Index est différent de Value ». Voir http://blog.adacore.com/testing-static-formal pour plus de détails. Le même lecteur attentif objectera ici que faire ce type de vérification après chaque appel peut devenir fastidieux et sérieusement impacter les performances de l'application. C'est que nous n'avons pas encore parlé des méthodes de vérification.

Vérification statique et preuve de programme

En première approximation, on note que le compilateur peut générer ou non des vérifications à la demande. Cette activation / désactivation est d'ailleurs très modulaire, elle peut s'appliquer au niveau d'un fichier, voir même offrir des politiques différentes au sein d'un même fichier. Le comportement par défaut d'un compilateur tel que GNAT (http://libre.adacore.com/tools/gnat-gpl-edition/) n'active d'ailleurs que les contrats simples (plage de valeur de type), les prédicats et autre pré et postconditions ne le seront qu'à la demande. De manière générale, la vérification de ces contraintes est particulièrement utile lors des phases de test unitaire ou de test d'intégration. Conserver ces protections après déploiement relève de choix stratégiques à étudier au cas par cas.

Mais le saint graal de la vérification n'est pas tant de détecter une erreur après son introduction, mais bien d'éviter qu'elle soit introduite en amont. Le test, bien positionné dans le cycle de développement, permet de le savoir très tôt, à espérer que l'échantillonnage soit suffisamment pertinent. Mais l'idéal est d'avoir un outil qui, par la seule force des mathématiques, puisse assurer de manière sure et terminale qu'un code est correct, ou sinon, pourquoi. Correct du point de vue de sa spécification il s'entend. Prouvé correct, le contrat a rempli son rôle et n'a plus d'utilité dans le code objet. Par hypothèse, il sera toujours vrai.

De nombreuses technologies existent pour se convaincre partiellement ou intégralement de la correction d'un contrat. Citons ici SPARK (http://libre.adacore.com/tools/spark-gpl-edition/), avec un cas d'exemple démontrant l'implémentation d'un jeu de Tetris formellement prouvé (http://blog.adacore.com/tetris-in-spark-on-arm-cortex-m4).

On notera humblement que la preuve de programme n'est cependant pas chose aisée. L'approche Ada permet d'exploiter la notation par contrat de manière pragmatique, en choisissant en fonction des compétences et des contraintes quels contrats écrire, et de quelle manière les vérifier.

Conclusion

Cette notion de programmation par contrat n'est pas nouvelle. Le langage Eiffel le proposait déjà à la fin des années 80. D'autres langages tels que C# s'y sont essayés, bien que ne disposant pas d'une sémantique de spécification logicielle bien définie. Ada est probablement unique dans le monde « mainstream » à proposer une sémantique bien définie, intégrée au langage, et permettant une approche hybride entre vérification dynamique et statique. Les outils sont disponibles. Entre deux sessions de débogage, pourquoi ne pas les essayer ?

Pourquoi créer un nouveau langage ?

J'ai démarré le développement du langage Oforth il y a plus de 15 ans. A l'époque, j'avais une certaine expertise dans les langages Assembleur, C, C++ et Smalltalk, et mes études m'ont amené à étudier dans le détail les problématiques de compilation.

Franck Bensusan
www.oforth.com

C'est à ce moment que j'ai découvert, par hasard, le langage Forth, un langage créé au début des années 70 par Charles Moore. Et ce fut un coup de cœur :

- Un langage qui utilisait la notation polonaise inversée, alors que j'avais passé toutes mes études sur des calculatrices HP, et encore plus de temps à essayer d'optimiser des programmes HP de 15 instructions, à décortiquer les programmes pour calculatrices de la revue « Jeux et Stratégie »...
- Un langage dynamique, qui compilait en une passe, au fil de l'eau, chose que je ne connaissais pas.
- Un langage dont la taille des programmes était bien inférieure à tout ce que j'avais vu jusque là. Plus compact que de l'assembleur ! Ce qui en faisait (ce qui en fait toujours, d'ailleurs) un langage fabuleux pour de la programmation embarquée. Par exemple, la sonde Philae qui s'est posée sur une comète embarque du code Forth dans plusieurs de ses applications temps réel.
- Un langage extensible avec la possibilité de créer de nouvelles syntaxes.

Ce fut vraiment une découverte, mais il me semblait que certains autres principes dataient vraiment, et la programmation orientée objet, les garbages collector, ... étaient maintenant courants. Autant le Forth était le langage idéal pour de l'embarqué, autant il me semblait qu'il avait un retard certain pour des applications desktop ou serveur. Pour être tout à fait honnête, n'ayant que peu d'expérience en programmation embarquée, je n'ai pas beaucoup programmé en Forth. L'idée d'un langage reprenant ces principes, mais en y incluant des fonctionnalités « modernes » s'est très rapidement imposée. La création du langage Oforth n'a donc pas été dictée par un rejet des autres langages. J'étais parfaitement à l'aise avec ceux que j'utilisais habituellement. Il s'agissait simplement de l'envie de voir ce que pourrait donner le Forth, dont j'étais impressionné par les principes, si on lui incluait des concepts modernes.

Comment cela se passe concrètement ?

Concrètement, c'est ... 15 ans ! Bien sûr, il y a eu de très longues périodes pendant lesquelles je n'ai absolument pas touché à Oforth. Je pense y avoir réellement travaillé pendant 5 ou 6 ans, et l'avoir réécrit en partie au moins 3 ou 4 fois.

Au début, il s'agit d'un hobby où les questions principales sont la syntaxe, le mariage entre une pile de données et les concepts objets, avec un focus particulier sur les performances. Comment gagner quelques nanosecondes lors de l'instanciation d'un objet ? Faire l'erreur d'écrire trop de lignes en assembleur, revenir en arrière, ...

Quelquefois, une simple idée vous fait réécrire 50% de votre code (le passage d'une compilation par tokens à une compilation native par exemple).

Puis, une fois que le langage commence à prendre forme, que la syntaxe se stabilise, et que les bases sont posées, d'autres questions d'un niveau plus élevé se posent :

- Essayer de limiter le plus possible le code natif au profit de code Oforth.
- Essayer d'intégrer au plus tôt UTF8 pour ne pas être bloqué par la suite.
- Faut-il intégrer la notion d'objet immutable ? Des concepts de programmation fonctionnelle ?
- Faut-il exposer les threads au programmeur ou pas ?
- Faut-il de la mémoire partagée et, si oui, comment la synchroniser ?

Tout cela passe par des jours d'investigation pour étudier les différentes approches possibles, identifier celles qui s'adaptent le mieux aux principes du Forth et, si nécessaire, en imaginer de nouvelles.

Par exemple, pour le modèle de parallélisme, la pile de donnée est un atout majeur dans un modèle de type tâche/channel puisqu'il n'est pas nécessaire de la sauvegarder lors des changements de contexte entre tâches. Ce type de modèle a donc été privilégié dans Oforth.

Enfin, la question la plus délicate : est-ce que le langage est arrivé à un niveau suffisant pour le rendre public ? Il a fallu des années avant d'avoir la conviction que le langage avait atteint une maturité suffisante. Et quand c'est le cas, on se lance...

Exemple d'un code Oforth :

```
func: pong(n, ch1, ch2) { | i | n loop: i [ ch2 send(ch1 receive) drop ] }

func: pingpong(n) {
| ch1 ch2 i |

  Channel new ->ch1
  Channel new ->ch2
  #[ pong(n, ch1, ch2) ] &
  n loop: i [ ch1 send(i) drop ch2 receive println ]
}
```


SWIFT : un langage en évolution constante depuis 12 mois !

La Worldwide Developer Conference (WWDC 2014) avait été l'occasion pour Apple d'annoncer en Juin dernier, la naissance de Swift, nouveau langage de développement pour ses plateformes iOS et OS X, destiné à remplacer à terme le populaire mais vieillissant Objective-C.

Stéphane CORDONNIER
http://blog.beecomedigital.com

Depuis près d'un an et en attendant la WWDC 2015 qui devrait être l'occasion pour Apple de faire la part belle à Swift, que s'est-il passé autour de ce langage qui a continué d'évoluer doucement mais sûrement au rythme des mises à jour successives de Xcode ?

Plusieurs mises à jour importantes et des performances accrues

Depuis la sortie de Xcode 6.0, première version de l'environnement de développement Apple à introduire le support de Swift 1.0, trois mises à jours ont été effectuées, dont deux d'entre elles concernaient directement Swift.

Xcode 6.1 fût l'occasion de découvrir Swift 1.1, qui était principalement centrée sur la correction de bugs du tout jeune langage. Cette première mise à jour du langage fût aussi l'occasion pour Apple de faire une revue de toutes ses APIs (Foundation, AppKit, UIKit, CoreData, WebKit...) et d'uniformiser leur interopérabilité avec Swift, notamment en ce qui concerne la gestion des « Optionals ».

Profitant de la sortie début Avril des mises à jour iOS 8.3 et Yosemite 10.10.3, Apple en a profité pour mettre à disposition de nouvelles versions de Xcode 6.3 et Swift 1.2. Cette dernière mise à jour corrige également nombre de bugs, mais a aussi été l'occasion de modifier le fonctionnement du langage, et surtout d'améliorer grandement ses performances.

Avec Swift 1.2, Apple a réalisé un gros travail d'optimisation au niveau du compilateur, tant en ce qui concerne les temps de compilation, que de la performance d'exécution de l'application une fois compilée.

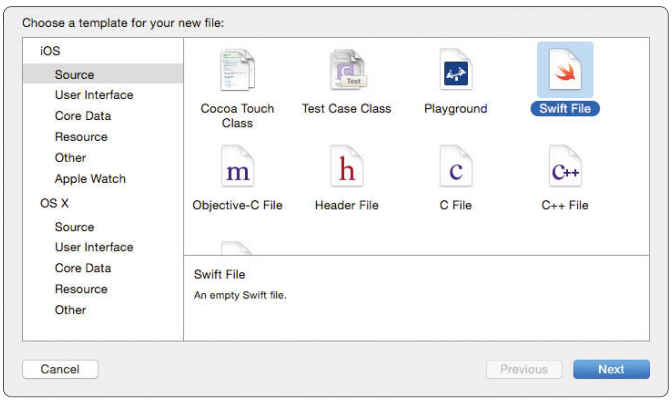
Pour les personnes ayant investi rapidement dans l'utilisation de Swift en créant des applications contenant plusieurs dizaines, voire centaines de fichiers/classes à compiler, il n'était pas rare de devoir attendre plusieurs dizaines de secondes, voire plusieurs minutes, à la compilation, même si un seul fichier avait été modifié entre deux compilations.

Désormais, le compilateur Swift est plus intelligent et un mode de compilation incrémentale a été introduit permettant théoriquement de ne recompiler que ce qui doit réellement l'être.

Ce mode fonctionne plutôt bien même si de temps à autre, une compilation totale de son application peut être relancée sans que l'on sache trop pourquoi.

Mais l'optimisation du compilateur s'est aussi beaucoup portée sur les performances à l'exécution lors d'utilisation de tableaux imbriqués, de calculs mathématiques manipulant beaucoup de nombres flottants, etc... Les applications actuelles s'exécutent généralement plus vite que les premières applications ayant pu être écrites avec Swift 1.0.

Une option nommée « Whole Module Optimization » a même été mise à disposition des développeurs dans les paramètres de « Build » de leur projet afin de pouvoir activer ou désactiver certaines optimisations en fonction des besoins (ex : désactivation en mode debug).



Des évolutions et améliorations du langage

Evidemment les mises à jour de Swift ne portent pas que sur les performances. Le langage a lui aussi continué d'évoluer au fil des mois en introduisant de nouvelles classes, de nouvelles fonctions, en modifiant et/ou améliorant certaines syntaxes existantes, etc...

Cela commence avec la notion de « Failable Initializer » qui permet dans une classe de pouvoir utiliser des constructeurs pouvant renvoyer des valeurs nulles.

```
extension Int {
  init?(fromString: String) {
    if let i = fromString.toInt() {
      self = i
    } else {
      return nil
    }
  }
}
```

L'extension de la classe « Int » ci-dessus permet ensuite d'utiliser cette syntaxe :

```
if let twentytwo = Int(fromString: "22") {
  println("the number is \(twentytwo)")
} else {
  println("not a number")
}
```

Concernant la déclaration de constantes à l'aide du mot clé « let », une modification importante a été effectuée. Auparavant, il était possible d'écrire ceci :

```
let someInts = [Int]()
someInts.append(3)
```

Désormais cette syntaxe ne fonctionne plus. En effet, à partir du moment où un tableau, un dictionnaire, etc... a été déclaré comme une constante, il devient « immutable » ce qui signifie qu'il n'est plus possible de modifier le nombre d'éléments qu'il contient. Pour pouvoir modifier le

tableau, il est nécessaire de le déclarer comme une variable :

```
var someInts = [Int]()
someInts.append(3)
```

Ensuite vient la possibilité d'effectuer des conversions de types. Il était jusqu'à présent possible d'écrire ceci afin d'effectuer une conversion pouvant échouer :

```
if let movie = item as? Movie {
  println("Movie: '\(movie.name)', dir. \(movie.director)")
}
```

Dans le cas où l'on sait que la conversion n'échouera pas et que le développeur veut forcer cette conversion, il est désormais possible d'écrire ceci (au risque de déclencher une erreur à l'exécution en cas d'échec) :

```
let movie = item as! Movie
println("Movie: '\(movie.name)', dir. \(movie.director)")
```

Pour continuer sur ce sujet, la conversion d'Objective-C vers Swift n'est désormais plus implicite, comme dans l'exemple ci-dessous :

```
func log(s: String) { println(x) }
let ns: NSString = "some NSString"
log(ns)    // Fails with the error 'NSString' is not convertible to 'String'
```

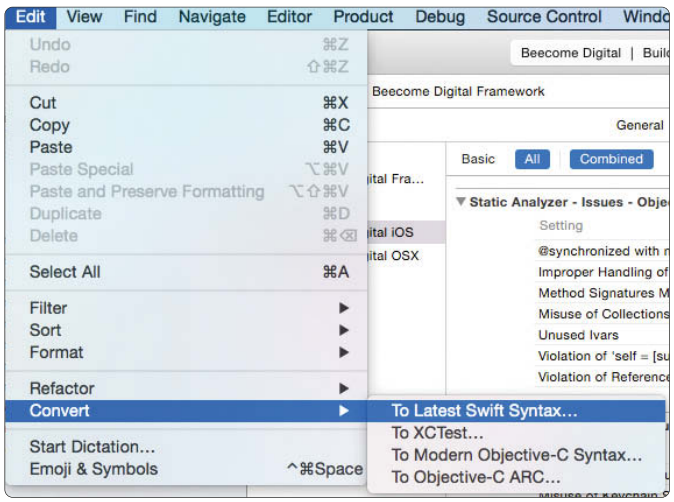
Pour que cet exemple fonctionne, il est désormais obligatoire d'écrire :

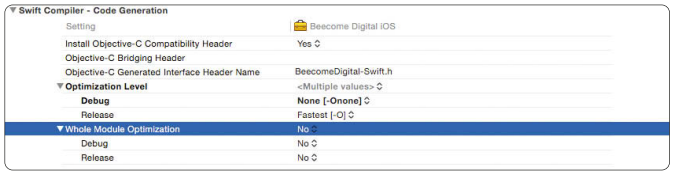
```
log(ns as String)
```

En revanche la conversion de Swift vers Objective-C peut toujours être utilisée de manière implicite. Le code ci-dessous fonctionne sans aucun problème :

```
func nsLog(ns: NSString) { println(ns) }
let s: String = "some String"
nsLog(s)
```

Dans les petites évolutions qui simplifient la vie, on retrouve la syntaxe ci-dessous. Elle permet, dans l'optique ou « items » serait nul, de pouvoir assigner la valeur zéro à la variable, plutôt que d'assigner la valeur « nil ».





La variable « itemCount » est dans ce cas de type « Int » au lieu de « Int? ».

```
var itemCount = items?.count ?? 0
```

Parlons également de l'imbrication de code comme on peut le voir ci-dessous :

```
if let a = foo() {
  if let b = bar() {
    if let c = baz() {
    }
  }
}
```

Cette syntaxe que l'on rencontre assez régulièrement quand on manipule des « Optionals », peut désormais être simplifiée de la manière suivante :

```
if let a = foo(), let b = bar(), let c = baz() {
}
```

Des évolutions également dans l'interopérabilité

Une part importante de l'engouement des développeurs pour Swift repose sur son intégration avec les APIs Objective-C existantes. Il est donc logique qu'Apple s'attarde à l'amélioration de l'interopérabilité entre les deux langages. C'est notamment le cas sur la gestion des « Optionals » avec l'introduction de nouveaux mots-clés, permettant à Swift de savoir comment est censé se comporter du code Objective-C lorsqu'il est utilisé avec Swift. C'est ce que l'on peut voir dans le code suivant, qui utilise la syntaxe « nonnull » et « nullable » :

```
-(void)registerNib:(nonnull UINib *)nib forCellReuseIdentifier:(nonnull NSString *)identifier;
-(nullable UITableViewCell *)cellForRowAtIndexPath:
 (nonnull NSIndexPath)indexPath;
@property (nonatomic, readwrite, retain, nullable) UIView *backgroundView;
```

Ce code une fois utilisé dans Swift, correspond aux déclarations suivantes, où l'on voit que les « nullable » ont été remplacés par des « Optionals », ce qui n'est pas le cas des « nonnull » :

```
func registerNib(nib: UINib, forCellReuseIdentifier identifier: String)
func cellForRowAtIndexPath(indexPath: NSIndexPath) -> UITableViewCell?
var backgroundView: UIView?
```

Vivement la WWDC 2015 !

Les évolutions de Swift 1.2 évoquées ne sont qu'un aperçu des plus marquantes/utiles.

D'autres évolutions du langage sont disponibles comme un nouveau type de données structurées (le « Set » équivalent du « NSSet » en Objective-C), de nouvelles fonctions dans la bibliothèque standard (flatMap(...) ou zip(...) par exemple), ou bien encore, un assistant dans Xcode permettant de migrer son ancien code Swift vers la dernière version.

Depuis près d'un an, les évolutions du langage se font doucement en fonction des retours que les développeurs font sur le forum officiel. Apple est en effet très à l'écoute des développeurs sur ce point.

Swift évolue dans le bon sens et le travail d'Apple pour faire évoluer ce nouveau langage est la preuve - s'il en fallait une - qu'il s'agit du langage du futur pour iOS et OSX.

Il est toutefois très probable que bon nombre d'améliorations de Swift et de l'outillage qui va autour, seront annoncées en primeur lors de la prochaine WWDC 2015. Rendez-vous en Juin prochain pour voir ce qu'il en est.

Le Bluetooth Low Energy dans les applications universelles

Le Bluetooth Low Energy (BLE dans la suite de cet article) est invariablement associé, à juste titre, aux objets connectés. C'est bien normal, le BLE est l'un des systèmes de communication phares d'une grande part des objets dits « connectés ».

Fig.1

Stéphane Sibué
Directeur technique chez **GUYZMO**
stephane.sibue@guyzmo.fr
www.guyzmo.fr

GUYZMO

Les objets connectés

Les objets connectés existent en fait depuis plus de 25 ans. Vous en côtoyez depuis très longtemps, chaque jour, sans même vous en rendre compte. Par exemple, votre distributeur de billets est un objet connecté, relié à la banque et complètement autonome, les stations météo au bord de certaines routes ou autoroutes aussi qui envoient leurs données collectées directement sans intervention extérieure. Plus proche de nous il y a aussi ces objets plus « modernes » tels que les bracelets, les montres, certaines cartes électroniques, qui ont eux aussi la capacité de collecter des données, et de les envoyer sans aucune intervention dans le cloud pour y être traitées, et j'en passe tellement nous sommes, sans vraiment le savoir, et depuis bien longtemps, entourés de ce type d'objets.

Aujourd'hui nous avons mieux conscience de leur existence car ils sont devenus plus petits et sont pour beaucoup plus intimes. On s'en sert pour évaluer notre activité physique, notre santé, ou pour « tracer » les déplacements de nos enfants. La miniaturisation et les progrès techniques ont permis à ces objets de faire partie intégrante de notre vie privée et nous les utilisons en connaissance de cause, pour notre usage exclusif Fig.1. Les objets connectés sont l'évolution naturelle des objets. Les objets ont d'abord été mécaniques, puis électriques, puis électroniques et maintenant ils sont communicants. Pour que ça marche, il faut 5 ingrédients de base :

- Des humains,
- Des objets,
- Des écrans pour les interactions,
- Le Cloud pour le stockage et l'intelligence,
- Et bien sûr de la connectivité.

Le Bluetooth Low Energy

Le BLE est une technique de transmission sans fil créée par Nokia sous forme d'un standard ouvert basé sur Bluetooth qu'il complète sans le remplacer. BLE correspond à la version 4 de Bluetooth.

Le BLE est un des protocoles clés de l'Internet des objets. Il est basé sur le Bluetooth « classique », avec un débit 10x moins important (1 Mb/s) mais consomme 10x moins. Autre point important, la latence de connexion et de transfert est très fortement réduite. Donc pour résumer, le BLE est certes moins rapide mais il consomme très peu et il est en mesure de restaurer très rapidement une connexion mise en sommeil, pour économiser de l'énergie. BLE est donc très adapté aux objets connectés car il prend en compte les importantes contraintes de la consommation énergétique.

SpecificationName	SpecificationType	AssignedNumber	SpecificationLevel
Alert Notification Service	org.bluetooth.service.alert_notification	0x1811	Adopted
Battery Service	org.bluetooth.service.battery_service	0x180F	Adopted
Blood Pressure	org.bluetooth.service.blood_pressure	0x1810	Adopted
Body Composition	org.bluetooth.service.body_composition	0x181B	Adopted
Bond Management	org.bluetooth.service.bond_management	0x181E	Adopted

Fig.2

Extrait de la liste des services normalisés

Les services et les caractéristiques

Le BLE est inclus dans la version 4 du standard Bluetooth. Sa grande particularité est d'être très fortement normalisé (voir le site Bluetooth Developer Portal : https://developer.bluetooth.org). Le BLE est organisé en services dont certains sont normalisés, chaque service possède 1 à n caractéristiques qui sont pour certaines normalisées aussi. Une caractéristique ressemble à une propriété du service, par exemple pour le service « Battery Service » on a la caractéristique « Battery Level» qui porte le niveau de batterie. Les services et les caractéristiques sont identifiés par des UUID (Universal Unique Identifier) qu'on a l'habitude d'appeler dans le vocabulaire Microsoft des GUID. Par exemple, le UUID du service « Battery Service » est « 0000**180F**-0000-1000-8000-00805F9B34FB » et comme c'est un service standardisé on ne parle que de la valeur « 0x180F » dans la documentation officielle, le reste de l'identifiant étant toujours le même. La caractéristique « Battery Level » a pour UUID « 0000**2A19**-0000-1000-8000-00805F9B34FB », et comme c'est une caractéristique standardisée on ne garde que « 0x2A19 » (même principe que pour le service) Fig.2.

Les caractéristiques possèdent des descripteurs. C'est-à-dire qu'une caractéristique est auto-documentée et que grâce à ses descripteurs vous êtes en mesure de savoir à quoi elle correspond, quel est son type, comment l'affi-

Service Characteristics

Overview	Properties	Security	Descriptors
Name: Battery Level **Description:** The Battery Level characteristic is read using the GATT Read Characteristic Value sub-procedure and returns the current battery level as a percentage from 0% to 100%; 0% represents a battery that is fully discharged, 100% represents a battery that is fully charged. **Type:** org.bluetooth.characteristic.battery_level **Requirement:** Mandatory	Property / Requirement Read: Mandatory Write: Excluded WriteWithoutResponse: Excluded SignedWrite: Excluded Notify: Optional Indicate: Excluded WritableAuxiliaries: Excluded Broadcast: Excluded ExtendedProperties:	None	Overview: **Name:** Characteristic Presentation Format; **Type:** org.bluetooth.descriptor.gatt.characteristic_presentation_format; **Requirement:** If_multiple_service_instances Permissions: Permission / Requirement: Read: Mandatory; Write: Excluded Overview: **Name:** Client Characteristic Configuration; **Type:** org.bluetooth.descriptor.gatt.client_characteristic_configuration; **Requirement:** If_notify_or_indicate_supported Permissions: Permission / Requirement: Read: Mandatory; Write: Mandatory

Fig.3

Détails d'une caractéristique avec ses descripteurs

cher, quelle est son unité, si on peut la lire et/ou l'écrire, est ce qu'elle permet les notifications, etc. Cela signifie que vous pouvez développer une application qui dans le pire des cas ne connaît rien des valeurs qu'elle doit lire ou écrire, elle devra le déterminer à l'aide des descripteurs Fig.3.

La normalisation des identifiants signifie aussi que votre application fonctionnera parfaitement avec tous les objets connectés exposant un service normalisé identique (ex toutes les ceintures cardiaques utilisant le service « Heart Rate » tel que défini dans le Bluetooth Portal).

Même si la normalisation est au cœur de la cohérence de BLE elle n'est pas obligatoire. Vous pouvez parfaitement créer votre service (avec un UUDI perso), composé de ses caractéristiques (normalisées ou non). Mais si votre produit est destiné au grand public et qu'il couvre des services normalisés alors il est préférable de se conformer à la norme.

Fig.4

Le SensorTag de Texas Instruments

Station Météo

Dans cet article je vous propose de réaliser une station météo avec un petit appareil bien sympathique, proposé par Texas Instruments, et qui répond au doux nom de « SensorTag ». Cette application sera développée en C# et sera de type « Universelle », c'est-à-dire qu'elle fonctionnera indifféremment sur Windows Phone 8.1 et sur Windows 8.1 Fig.4.

Le SensorTag est un démonstrateur. Il permet à Texas Instruments de faire tester sa technologie (capteurs, modules de communication, etc.) d'une manière simple et peu chère. Le SensorTag (dont vous trouverez une fiche descriptive ici : http://www.ti.com/tool/cc2541dk-sensor) est bardé de capteurs et communique en BLE (ça tombe bien me direz-vous). Il possède en standard les capteurs suivants :

- Capteur de température ambiante
- Capteur de température infra-rouge
- Capteur de pression atmosphérique
- Capteur d'humidité
- Accéléromètre
- Gyroscope
- Magnétomètre
- 2 boutons poussoirs

Avec les capteurs de température, de pression et d'humidité on peut assez facilement créer une petite station météo, du moins un baromètre amélioré. Le capteur de température ambiante est théoriquement celui qu'il faudrait utiliser dans une station météo. Ce capteur est assez lent à rendre compte d'un changement de température, ce qui est assez frustrant pour faire une démo. Dans cet exemple nous allons utiliser le capteur de température infra-rouge qui normalement sert à déterminer la température d'un objet distant. Ce capteur est très rapide et permet de « voir » les changements de température dans l'application de manière rapide. Idéalement, il faudrait remplacer l'utilisation de ce capteur par celui de la température ambiante si vous souhaitez utiliser dans de véritables conditions cette petite station météo.

> **Important :** *Sous Windows Phone et sous Windows, vous devez impérativement appairer les devices BLE avant de pouvoir les utiliser. Sur d'autres plateformes (iOS, Android) il est possible de partir à la découverte des devices BLE sans les avoir appairés au préalable, et ensuite de s'y connecter à la volée. Sous Windows Phone et Windows c'est pour le moment impossible. Cette situation devrait changer avec l'arrivée de Windows 10.*

Services et caractéristiques utilisées

Voici un tableau des services et caractéristiques utilisés. Vous remarquerez que les UUID ne sont pas standards et ont été définis par Texas Instruments. La documentation du SensorTag fournit toutes les informations à ce sujet. Les 3 capteurs que nous allons utiliser fonctionnent avec 2 caractéristiques. La 1ère (Data) permet de lire la donnée (température, pression, humidité), la seconde (Config) permet d'activer ou désactiver le capteur (important pour une bonne gestion de l'énergie). Le service de pression (Barometer Service) utilise une 3ème caractéristique permettant la calibration du capteur.

Service ou caractéristique	UUID
IR Temperature Service	F000AA00-0451-4000-B000-00000000000
Data	F000AA01-0451-4000-B000-00000000000
Config	F000AA02-0451-4000-B000-00000000000
Humidity Service	F000AA20-0451-4000-B000-00000000000
Data	F000AA21-0451-4000-B000-00000000000
Config	F000AA22-0451-4000-B000-00000000000
Barometer Service	F000AA40-0451-4000-B000-00000000000
Data	F000AA41-0451-4000-B000-00000000000
Config	F000AA42-0451-4000-B000-00000000000
Calibration	F000AA43-0451-4000-B000-00000000000

L'application Station Météo

Application universelle, qui affichera :

- La température en °C
- La pression atmosphérique en hPa (hecto pascal)
- Le degré d'humidité en %

La pression atmosphérique permet de déterminer un type de temps, comme on le fait avec un bon vieux baromètre. On peut utiliser les formules suivantes pour déterminer le type de temps :

Pression en hPa	Type de temps
Inférieur à 980	Tempête
De 980 à 1000	Pluie ou vent
De 1001 à 1030	Variable
De 1031 à 1050	Beau temps
Supérieur à 1050	Très sec

L'écran sera très simple avec les 3 informations de base affichées (température, pression, humidité) ainsi que le type de temps qui sera affiché sous la forme d'un pictogramme et d'un texte en clair (Variable, Tempête, etc) Fig.5 et 6. L'écran Windows Phone 8.1 sera affiché en portrait alors que celui pour Windows 8.1 sera affiché en paysage. Ce sera la seule différence

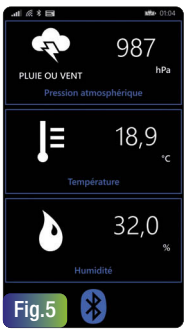


Fig.5

Ecran Windows Phone 8.1

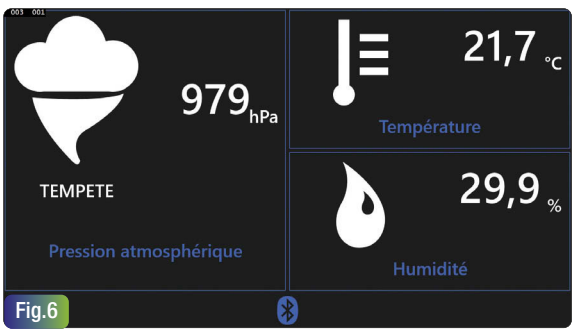


Fig.6

Ecran Windows 8.1

entre les 2 versions de cette application universelle. En gros 95% du code est identique, c'est le grand intérêt des applications universelles.

Le projet Station Météo

Cette application étant de type « application universelle », la majorité du code est placée dans le projet « Shared ». Seul le fichier « MainPage.xaml » est propre à chaque plateforme. Leur différence étant due principalement à l'orientation de l'écran Fig.7.

Définir les services utilisés dans l'application

La 1ère chose à faire, et à ne surtout pas oublier, est de définir dans le fichier « Package.appxmanifest » de chaque plateforme les services BLE qui seront utilisés par l'application. Sans cela, votre application se comportera comme si elle ne « voyait » pas les services BLE concernés alors qu'ils sont disponibles et actifs. Vous devez définir les services de la manière suivante : Fig.8.

> **Note :** *Dans la documentation WinRT vous verrez qu'il est aussi possible de définir les services normalisés (on n'en utilise pas dans ce projet) avec leur nom plutôt que leur UUID.*

L'espace de nom contenant tous les objets permettant de gérer le BLE est : **Windows.Devices.Bluetooth.GenericAttributeProfile**

Connexion à un service

Pour travailler avec un service, il faut dans un 1er temps s'y connecter. Pour le faire on utilise un filtre qui permettra de récupérer le service désiré. Ce filtre permet de ne sélectionner que les devices appairés porteurs de ce service. Dans notre cas nous allons donc filtrer sur un des services du SensorTag dont nous avons besoin (par exemple la température) et vérifier que le ou les devices supportant ce service sont bien de type SensorTag. Au final nous retiendrons le 1er device répondant à tous ces critères. Une fois cette sélection faite, on obtiendra un objet de type **GattDeviceService** qui correspond à un device (un SensorTag dans notre cas) branché sur le service souhaité. Voici un exemple de fonction permettant de se connecter à un service particulier d'un SensorTag :

```
protected async Task<GattDeviceService> GetDeviceService(Guid wService)
{
  var wFilter = GattDeviceService.GetDeviceSelectorFromUuid(wService);
  var wDevices = await DeviceInformation.FindAllAsync(wFilter);
  foreach (var wDevice in wDevices)
  {
    if (wDevice.Name == "TI BLE Sensor Tag")
    {
      // On a trouvé un SensorTag
      return await GattDeviceService.FromIdAsync(wDevice.Id);
    }
  }
  return null;
}
```

On lui passe l'UUID d'un service (sous la forme d'un GUID en .NET) et il nous retourne un objet de type **GattDeviceService** si le service demandé est bien disponible sur un SensorTag connecté. Notez que l'application du filtre et la récupération du **GattDeviceService** sont des fonctions asynchrones. Pour utiliser les fonctions BLE il faut donc connaître le fonctionnement du couple async/await.

Connexion à une caractéristique

A partir du **GattDeviceService** récupéré et pointant sur le service qui nous intéresse, on récupère une caractéristique de ce service en utilisant son UUID. La fonction de récupération de caractéristique retourne une collection dont on n'utilisera que le 1er élément. Une collection vide signifiant que la caractéristique demandée n'est pas implémentée par le service. On obtient alors un objet de type **GattCharacteristic** correspondant à la caractéristique souhaitée.
Voici un exemple de fonction permettant de récupérer une caractéristique dont le service et l'UUID sont passés en paramètres :

```
protected GattCharacteristic GetCharacteristic(GattDeviceService deviceService, Guid
 characteristicUUID)
{
  var characteristics = deviceService.GetCharacteristics(characteristicUUID);
  if (characteristics.Count > 0)
  {
    return characteristics[0];
  }
  else
  {
    return null;
  }
}
```

Ecriture d'une caractéristique

On ne peut écrire qu'un train d'octets qui doit être fourni sous la forme d'un buffer de type **IBuffer**. Cela signifie qu'il faut en passer par la conversion de données de base vers leur version binaire pour lire ou écrire des données.
Pour l'écriture d'une caractéristique on utilise généralement un objet de type **DataWriter** qui s'occupe de générer le IBuffer à partir des données qu'on lui a fournies. C'est la méthode asynchrone **WriteValueAsync** qui se charge d'écrire les données dans la caractéristique.
Voici un exemple de code qui écrit la valeur 1 dans une caractéristique passée en paramètre. Cette valeur est de type **Byte**. Dans le SensorTag c'est

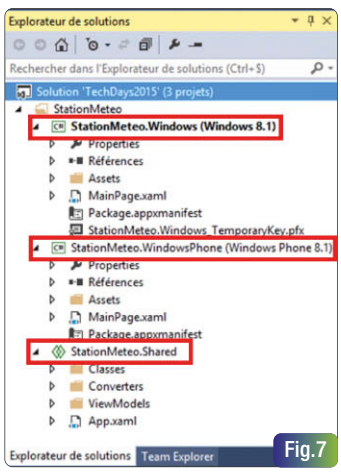


Fig.7 *Projet StationMeteo*

```
<Capabilities>
  <Capability Name="internetClient" />
  <m2:DeviceCapability Name="bluetooth.genericAttributeProfile">
    <m2:Device Id="any">
      <m2:Function Type="serviceId:F000AA00-0451-4000-B000-000000000000" />
      <m2:Function Type="serviceId:F000AA20-0451-4000-B000-000000000000" />
      <m2:Function Type="serviceId:F000AA40-0451-4000-B000-000000000000" />
    </m2:Device>
  </m2:DeviceCapability>
</Capabilities>
```

Fig.8 *Les 3 services utilisés dans l'application sont définis dans le fichier Package.appxmanifest*

ce qu'on fait quand on veut activer un capteur, on écrit 1 dans la caractéristique de configuration du service gérant le capteur qu'on veut activer :

```
public async Task<bool> StartSensor(GattCharacteristic characteristic)
{
  if (characteristic != null)
  {
    DataWriter writer = new DataWriter();
    writer.WriteByte(1);
    var status = await characteristic.WriteValueAsync(writer.DetachBuffer());
    writer.Dispose();

    return (status == GattCommunicationStatus.Success);
  }
  return false;
}
```

Cette fonction retourne **True** si l'écriture s'est bien passée et **False** dans le cas contraire. Elle est asynchrone. Le **DataWriter** permet d'écrire des données dans beaucoup de formats, les unes à la suite des autres, dans un buffer de sortie. Le buffer de sortie est récupéré par sa méthode **DetachBuffer()**.

Lecture d'une caractéristique

De la même manière qu'on ne peut écrire qu'un train d'octets, on ne peut lire qu'un train d'octets (logique me direz-vous). Ce train d'octets est aussi fourni sous la forme d'un buffer de type **IBuffer**. On peut s'aider d'un objet de type **DataReader** pour « décoder » les valeurs présentes dans le buffer de lecture.

```
public async Task<Byte> GetByteValue(GattCharacteristic characteristic)
{
  if (characteristic != null)
  {
    var result = await characteristic.ReadValueAsync(BluetoothCacheMode.Uncached);

    if (result.Status == GattCommunicationStatus.Success)
    {
      var reader = DataReader.FromBuffer(result.Value);
      byte b = reader.ReadByte();
      reader.Dispose();
      return b;
    }
  }
  return 0;
}
```

Cette fonction, très simple, lit une caractéristique passée en paramètre retournant une valeur de type **Byte**.

Activation des notifications

Le système peut déclencher un événement lorsque la valeur d'une caractéristique change. Pour que ce soit possible il faut que la caractéristique le permette. On le sait en lisant la documentation du serveur ou alors on peut aussi le vérifier grâce aux descripteurs de cette caractéristique.
Il ne suffit pas de s'abonner à l'événement **ValueChanged** d'une caractéristique pour être mis au courant d'un changement, il faut aussi activer coté caractéristique les notifications (toujours en réponse au problème de gestion de l'énergie). Voici un exemple de code qui se « branche » sur les notifications d'une caractéristique pointée par la variable pDataCharacteristic :

```
await pDataCharacteristic.WriteClientCharacteristicConfigurationDescritorAsnyc
(GattClientCharacteristicConfigurationDescriptorValue.Notify);

pDataCharacteristic.ValueChanged += pDataCharacteristic_ValueChanged;
```

Et plus loin dans le code :

```
private void pDataCharacteristic_ValueChanged(GattCharacteristic sender, GattValueChanged
EventArgs args)
{
  // Code déclenché en cas de nouvelle valeur (callback)
  // args.CharacteristicValue contient le train d'octets de la nouvelle valeur
}
```

Conclusion de cette 1ère partie

L'utilisation du BLE dans une application Windows Universelle est assez simple à mettre en œuvre. Il faut tout de même au préalable bien comprendre comment est organisé le protocole BLE et comment fonctionnent toutes les fonctions WinRT dédiées à cette tâche. Dans la 2ème partie de cet article nous allons voir comment chaque capteur est géré par des objets spécialisés rendant leur utilisation dans n'importe quelle application très simple. A la fin de cette 2ème partie vous aurez tous les outils pour utiliser le BLE dans vos applications, les yeux fermés. ⊞

Faites de la musique avec le plugin VST

Quand on commence à s'intéresser à la musique assistée par ordinateur (M.A.O), on est très vite confronté à un problème de taille : comment puis-je éditer et modifier mes pistes pour obtenir le son que je désire ?

Patrick-André Marendat
Softfluent

On commence donc à se renseigner sur les différentes techniques et technologies qui existent, et on se rend compte que des standards existent (ouf, on a eu chaud !). Parmi eux, il en existe un qui va nous intéresser en particulier pour cet article, à savoir, le plugin VST. Non content d'apprendre ce qu'est un plugin VST, nous allons apprendre à faire le notre en utilisant VST.NET. Vous serez guidé pas à pas tout au long de cette fantastique aventure qu'est la création d'un plugin !
Commençons donc par les notions basiques qu'il faut connaître pour être à l'aise dans notre création. Un plugin VST est une interface qui sert à intégrer soit des instruments virtuels (type Addictive Drums ou BFD), soit des effets audios (type délais, chorus) à l'intérieur de notre séquenceur favori (type Reaper, Logic Pro, Cubase ou Pro Tools) Fig.1.
En 1996, une société nommée Steinberg (bien connue pour son séquenceur Cubase), dévoile le premier SDK de plugin VST. En 1999, ce sont les plugins d'instuments virtuels qui sont ajoutés à cette technologie.

Qu'est-ce que VST.NET et pourquoi l'utiliser ?

VST.NET n'est autre qu'une passerelle directe entre les fonctions C++ du SDK et le monde .NET. Utiliser cette technologie permet d'appréhender plus facilement la programmation de plugins VST qu'en utilisant le SDK natif. De plus, VST.NET est utilisable avec n'importe quel langage de .NET. Enfin, VST.NET se veut plus simple d'accès et plus clair grâce à une bonne documentation.

LE VIF DU SUJET : COMMENT RÉALISER UN PLUGIN VST DE DÉLAI AVEC VST.NET ?

Afin de mieux vous accompagner, nous n'aborderons que les parties techniques essentielles à la compréhension de l'article. Cependant, le code source complet est disponible sur GitHub : https://github.com/pmasf/VSTNETExemple .

Se procurer et explorer VST.NET.

Il suffit de se rendre sur http://vstnet.codeplex.com/ et de télécharger la dernière version de VST.NET.
En explorant le contenu téléchargeable, on voit qu'il contient les différentes librairies constituant VST.NET ainsi qu'une aide.
Les différentes dlls sont :

- Jacobi.Vst.Interop.dll : permet de faire le lien entre le monde non managé du séquenceur et le monde managé du plugin,
- Jacobi.Vst.Core.dll : regroupe toutes les fonctionnalités de base de VST.NET,
- Jacobi.Vst.Framework.dll : se présente comme une surcouche à la dll précédente et expose sous forme d'interfaces des regroupements de fonctionnalités afin de faciliter le travail.

L'aide se présente sous la forme d'un fichier très complet de documentation. Tous les concepts de base sont présents, et tout ce qu'il y a à savoir sur VST.NET est disponible dans cette aide. C'est une vraie

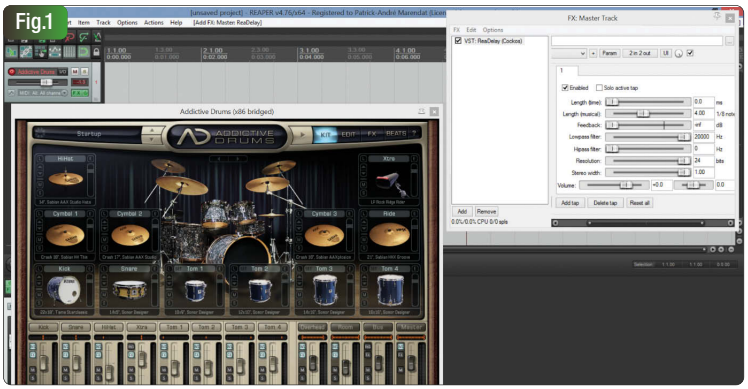


Fig.1

une mine d'or pour ceux qui veulent vraiment comprendre comment fonctionne VST.NET.
Attention cependant, lorsque j'ai suivi les différentes parties de l'aide, je suis tombé sur des exemples de code qui ne compilaient pas.
Comme toujours, il vaut mieux comprendre que recopier sans réfléchir !

Configurer son projet Visual Studio

Nous allons maintenant voir comment configurer notre projet Visual Studio afin de pouvoir utiliser dans les meilleures conditions VST.NET.
La première chose à savoir est que notre rendu final sera une dll. Il faut donc choisir le type de projet Class Library.
Pour que notre plugin soit reconnu par notre logiciel hôte (le séquenceur), il faut que notre plugin suive une certaine convention de nommage. Afin de rendre plus automatique les renommages des dlls produites en fin de build, il est bien de suivre cette étape, qui reste optionnelle, et qui consiste à éxécuter des instructions post-build.
Pour ajouter des instruction post-build, il suffit d'aller dans les propriétés de votre projet, de cliquer sur l'onglet Build Events et de mettre les lignes suivantes dans le champ Post-Build :

```
copy "$(TargetPath)" "$(SolutionDir)_SharedAssemblies\$(TargetName).net.dll"
copy "$(SolutionDir)_SharedAssemblies\Jacobi.Vst.Interop.dll" "$(SolutionDir)_Shared
Assemblies\$(TargetName).dll"
```

Ces lignes servent uniquement à copier et renommer les dlls produites après notre build. Pour que les instruction post-build ne plantent pas quand on build, il faudra créer un dossier _SharedAssemblies au niveau de la racine de votre solution Visual Studio.
Dans ce dossier, vous devrez copier la dll Jacobi.Vst.Interop.dll qui sera ensuite renommée en NOM_DE_VOTRE_PLUGIN.dll.
Il faut, bien entendu, ajouter à votre projet les références aux deux dlls suivantes :

- Jacobi.Vst.Core.dll
- Jacobi.Vst.Framework.dll

Pour ce faire, faites un clic droit sur References dans l'arborescence du Solution Explorer et Add Reference.

Création d'un plugin "vide"

Le premier plugin que nous allons créer n'est pas le plugin le plus sexy

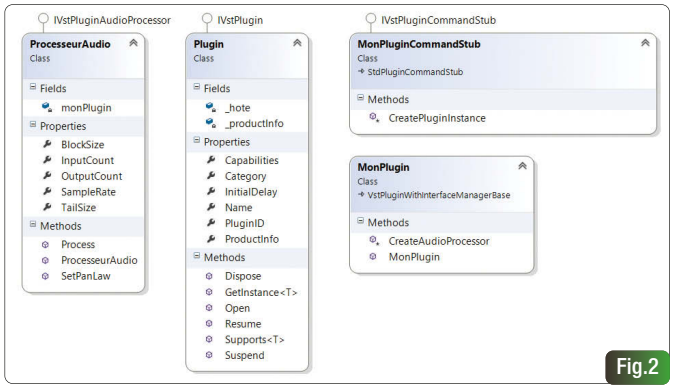


Fig.2

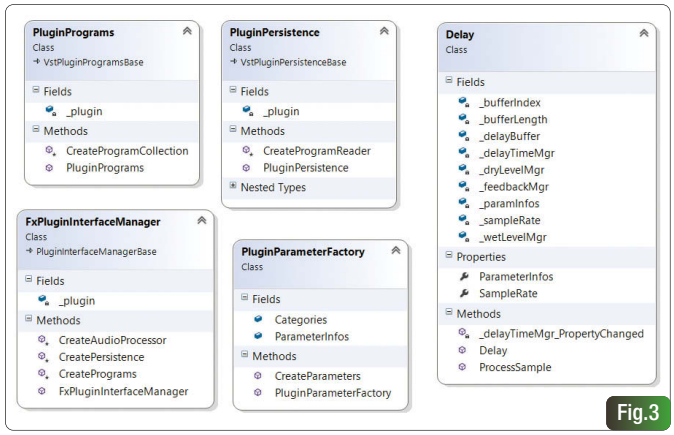


Fig.3

du monde, à vrai dire il ne fait que reproduire ce qu'il a en entrée vers sa sortie. Le but ici est de comprendre comment fonctionne VST.NET. Nous verrons ensuite un plugin légèrement plus complexe.
Afin de créer notre premier plugin « vide », il nous faut seulement quatre classes qui implémenteront des interfaces du Framework ou qui dériveront de classes du Framework.
Voici le diagramme des classes de mon projet **Fig.2**.
Ainsi que les implémentations commentées de chaque classe.

MonPlugin.cs

```
using Jacobi.Vst.Core;
using Jacobi.Vst.Framework;
using Jacobi.Vst.Framework.Plugin;


namespace VST001
{
  class MonPlugin : VstPluginWithInterfaceManagerBase
  {
    // Constructeur de MonPlugin
    public MonPlugin()
      : base("Mon Super Fun Plugin",
        new VstProductInfo("SFPlugin", "PAFactory", 1000),
        VstPluginCategory.RoomFx,
        VstPluginCapabilities.NoSoundInStop,
        0,
        0)  // Il faut mettre un identifiant unique ici !
    {
    }

    // Le code complet se trouve ici : https://github.com/pmasf/VSTNETExemple
  }
}
```

ProcesseurAudio.cs

```
using Jacobi.Vst.Core;
using Jacobi.Vst.Framework;


namespace VST001
{
  class ProcesseurAudio : IVstPluginAudioProcessor
  {
    /// <summary>
    /// Méthode qui effectue le traitement
    /// </summary>
    /// <param name="inputs">Entrées</param>
    /// <param name="outputs">Sorties</param>
    public void Process(VstAudioBuffer[] inputs, VstAudioBuffer[] outputs)
    {
      // On affecte à nos variables de traitement la première entrée et la première sortie (canal mono 1)
      VstAudioBuffer input = inputs[0];
      VstAudioBuffer output = outputs[0];

      // Pour chaque échantillon en entrée on recopie le signal en sortie.
      for (int index = 0; index < output.SampleCount; index++)
      {
        output[index] = input[index];
      }

      // On effectue le même traitement avec la seconde entrée et la seconde sortie (canal mono 2)
      input = inputs[1];
      output = outputs[1];

      for (int index = 0; index < output.SampleCount; index++)
      {
        output[index] = input[index];
      }
    }
    // Le code complet se trouve ici : https://github.com/pmasf/VSTNETExemple
  }

}
```

Ajouter de l'intelligence à son plugin.

A ce moment précis vous devriez être en possession de votre premier plugin fait maison ! Il n'est malheureusement pas très intelligent... Nous allons remédier à cela **Fig.3**.

Delay.cs

```
using Jacobi.Vst.Framework;
using System.ComponentModel;

namespace VST001
{
  internal class Delay
  {

    /// <summary>
    /// Constructeur
    /// </summary>
    public Delay()
```

```
    {
        // On instancie une nouvelle collection d'informations pour les paramètres du délai
        _paramInfos = new VstParameterInfoCollection();

        #region Initialize Parameters

        // Paramètres du temps du délai
        VstParameterInfo paramInfo = new VstParameterInfo();
        paramInfo.CanBeAutomated = true;
        paramInfo.Name = "dt";
        paramInfo.Label = "Delay Time";
        paramInfo.ShortLabel = "T-Dly:";
        paramInfo.MinInteger = 0;
        paramInfo.MaxInteger = 1000;
        paramInfo.LargeStepFloat = 100.0f;
        paramInfo.SmallStepFloat = 1.0f;
        paramInfo.StepFloat = 10.0f;
        paramInfo.DefaultValue = 200f;
        _delayTimeMgr = new VstParameterManager(paramInfo);
        VstParameterNormalizationInfo.AttachTo(paramInfo);

        _paramInfos.Add(paramInfo);

        #endregion

        // On s'abonne à l'event de changement de la propriété de temps du délai
        _delayTimeMgr.PropertyChanged += new PropertyChangedEventHandler(_delayTimeMgr_
PropertyChanged);
    }

    /// <summary>
    /// Permet de traiter chaque échantillon
    /// </summary>
    public float ProcessSample(float sample)
    {
        if (_delayBuffer == null) return sample;

        // Traitement de la sortie
        float output = (_dryLevelMgr.CurrentValue * sample) + (_wetLevelMgr.CurrentValue * _delay
Buffer[_bufferIndex]);

        // Traitement du buffer du délai
        _delayBuffer[_bufferIndex] = sample + (_feedbackMgr.CurrentValue * _delayBuffer[_bufferIndex]);

        _bufferIndex++;

        // Gestion de la position du buffer
        if (_bufferIndex >= _bufferLength)
        {
            _bufferIndex = 0;
        }

        return output;
    }
 // Le code complet se trouve ici : https://github.com/pmasf/VSTNETExemple
  }
}
```

MonPlugin.cs (V2)

```
using Jacobi.Vst.Core;
using Jacobi.Vst.Framework;
using Jacobi.Vst.Framework.Plugin;


namespace VST001
{
  class MonPlugin : VstPluginWithInterfaceManagerBase
  {
    private FxPluginInterfaceManager _intfMgr;
    /// <summary>
    /// Paramètres du plugin
    /// </summary>
    public PluginParameterFactory ParameterFactory { get; private set; }

    public MonPlugin()
      : base("Mon Super Fun Plugin",
        new VstProductInfo("SFPlugin", "PAFactory", 1000),
        VstPluginCategory.RoomFx,
        VstPluginCapabilities.NoSoundInStop,
        0,
        0)
    {

      _intfMgr = new FxPluginInterfaceManager(this);
      ParameterFactory = new PluginParameterFactory();
      ProcesseurAudio audioProcessor = _intfMgr.GetInstance<ProcesseurAudio>();
      // On ajoute le délai aux paramètres du plugin
      ParameterFactory.ParameterInfos.AddRange(audioProcessor.Delay.ParameterInfos);
    }
 // Le code complet se trouve ici : https://github.com/pmasf/VSTNETExemple
  }
}
```

ProcesseurAudio.cs (V2)

Le code complet se trouve ici : https://github.com/pmasf/VSTNETExemple

Seules les classes MonPluginCommandStub et Plugin ne changent pas.

Comment utiliser son plugin dans Reaper ?

Ça y est, vous avez enfin votre premier plugin sous la main. Reste maintenant à l'installer correctement dans votre séquenceur préféré et à tester le tout !

Reaper possède un dossier de plugins au chemin suivant C:\Program Files\REAPER (x64)\Plugins\FX .

Il suffit d'y coller vos dlls (Jacobi.Vst.Core.dll, Jacobi.Vst.Framework.dll, VOTRE_NOM_DE_PLUGIN.net.dll et VOTRE_NOM_DE_PLUGIN.dll)

Les nouveaux plugins sont scannés par Reaper automatiquement mais vous pouvez le faire aussi manuellement. Si vous voulez programmer votre propre plugin, il vous faudra tester dans votre séquenceur ce dernier. Dans ce cas il faudra supprimer les versions précédentes de votre plugin pour pouvoir voir la nouvelle.

Une fois le plugin détecté il suffit de le tester !

Pour aller plus loin...

Nous avons couvert ici les grandes lignes de la création de plugin via VST.NET. N'oubliez pas de vous rendre sur le dépot GitHub pour avoir le code source complet : https://github.com/pmasf/VSTNETExemple .

Les wearables à l'heure des montres connectées

Les wearables sont ces dispositifs électroniques et informatiques miniatures portés sur le corps. Ils sont capables de nous envoyer des informations ou de quantifier nos activités physiques grâce à des capteurs intégrés. Ils peuvent se connecter directement à Internet ou en s'appairant à un smartphone. Les wearables peuvent être portés sous la forme d'un accessoire (e.g., collier, bracelets, montres, bijoux, lunettes) ou intégrés dans les vêtements (e.g., T-shirt). Dans cet article, nous allons introduire la technologie des montres connectées.

Sameh Ben Fredj, *consultante chez* **Xebia**

Thomas Guerin, *consultant chez* **Xebia**

Durant ces 3 dernières années, le marché des montres connectées (Smartwatch en anglais) a particulièrement connu une forte progression. Selon le cabinet d'études indépendant *SmartWatch Group*, les ventes des montres connectées sont passées de 3,1 millions à 6,8 millions en 2014. Selon *NextMarket*, les prévisions de ventes sont à la hausse avec plus de 100 millions de montres connectées vendues d'ici 2017. Enfin, Selon le cabinet *GfK*, les ventes de "Smartwatch" devraient devenir le premier marché des objets connectés Fig.1.
Ce succès connu par les montres connectées nous amène à nous intéresser à cette technologie pour en comprendre son fonctionnement.

Une montre connectée, c'est quoi ?

Il n'existe pas de définition unique de la montre connectée. Cependant, la plus fréquente est celle donnée par le groupe d'étude Smartwatch Group qui définit la montre connectée comme étant un objet qui répond aux critères suivants :

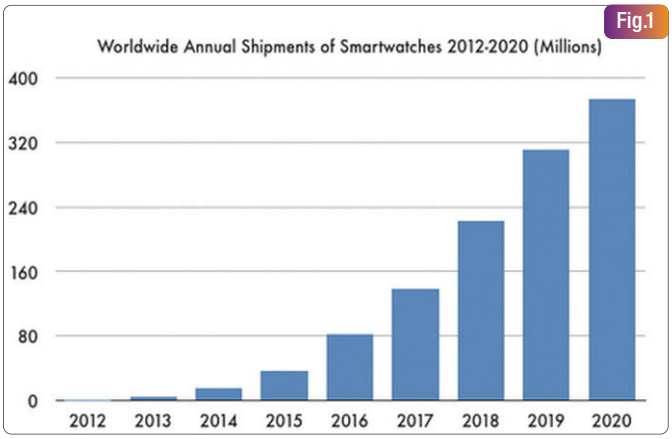
- Portée au poignet,
- Affiche l'heure,
- Dispose d'une connectivité sans fil (e.g., bluethooth, wifi).

Ce 3ème point, le plus important, représente l'intelligence même de la montre connectée par rapport à une montre traditionnelle. Sa capacité à se connecter à des réseaux sans fil lui apporte des nouvelles fonctionnalités.

Evolution

Avec le développement des montres électroniques, les premières montres connectées sont apparues au début des années **2000** avec IBM Watch Pad qui implantait le Bluetooth, un accéléromètre et affichait l'heure. En **2006**, Microsoft a dévoilé son projet SPOT (Smart Personal Objects Technology) proposant des montres connectées au réseau de la radio FM aux US et au Canada. Elles étaient capables de délivrer l'heure exacte selon la zone géographique, la musique locale et des informations sur la circulation. Le projet ne fut pas un grand succès. En **2007,** Sony Ericsson a sorti la montre MBW-150 capable de s'appairer à un téléphone. En **2009**, Samsung a sorti le modèle S9110 capable de fonctionner comme un téléphone. Cependant, la plus connue des Smartwatch nouvelle génération est probablement Pebble sortie en **2012 :** ce projet a rassemblé 10 millions de dollars de financement sur kickstarter. En **2013**, les montres Samsung Galaxy Gear, fonctionnant sur Android, sont arrivées sur le marché. L'arrivée de l'OS Android Wear en **2014** permet d'améliorer les performances des montres connectées en termes d'autonomie et d'usage comme la montre de Motorola, la Moto 360. L'année **2015** sera marquée par l'arrivée de la iWatch d'Apple, les montres connectées haut de gamme Fig.2.

Fig.1



Evolution annuelle des ventes de Smartwatch *(Source : NextMarket, 2013)*

Classification

Il existe aujourd'hui une multitude de montres connectées sur le marché : des Pebbles aux montres LG, Samsung, Motorola ou Apple. Une première classification est possible selon leur dépendance au téléphone. On peut donc distinguer deux grandes familles :

Les montres connectées dépendantes du téléphone :

Dans ce cas, la montre a besoin de s'appairer à un téléphone de même OS pour être fonctionnelle. Elle se connecte au téléphone

Fig.2

Évolution des montres connectées

essentiellement en Bluetooth. Dans cette famille, nous allons essentiellement trouver les Apple Watch (iWatch) et les montres fonctionnant sur Android Wear (Samsung, HTC).

Les montres autonomes

Le téléphone n'est pas nécessaire pour la faire fonctionner. Nous pouvons citer par exemple la Samsung Galaxy Gear ou l'Omate TrueSmart. Cette famille de montres est cependant gourmande en énergie, elle nécessite une carte SIM pour fonctionner et peut être volumineuse.

On peut aussi effectuer une classification plus fine des montres sur la base de leurs caractéristiques techniques : la taille et technologie de l'écran, sa compatibilité avec les différents OS (Android, iOS), son autonomie, sa connectivité, etc.

Les tendances du marché montrent que, actuellement, les montres dépendantes au téléphone ont plus de succès que les montres complètement indépendantes. En effet, avec la multiplication des smartphones, les montres connectées s'affichent comme leur compagnon idéal pour de nouveaux usages.

Usages

Les usages des montres connectées sont multiples, on liste essentiellement :

L'affichage des notifications et possibilités d'y répondre rapidement

Tout au long de la journée on reçoit plein de notifications (SMS, email, réseaux sociaux) et d'appels sur notre téléphone. Aussi, on peut avoir besoin de trouver son chemin, choisir sa musique, utiliser le GPS ou consulter la météo. Toutes ces requêtes exigent de sortir son téléphone et de le déverrouiller à chaque fois. La montre connectée permet d'accéder rapidement à l'information directement en jetant un coup d'oeil sur le poignet et sans avoir à toucher son smartphone ou répondre directement aux appels reçus à partir de la montre.

Le suivi des activités physiques

La montre connectée peut aussi être un bracelet connecté qui embarque des capteurs et mesure l'activité physique. Il existe des usages dans le domaine du sport et du bien-être avec un suivi des performances et des efforts physiques : mesures de vitesses, nombre de kilomètres parcourus, nombre de calories dépensées. D'autres usages existent aussi dans le domaine de la santé avec, par exemple, la mesure du rythme cardiaque et la tension artérielle.

COMMENT FONCTIONNENT LES MONTRES SOUS ANDROID WEAR ?

Avec 6 gammes de montres connectées, les montres sous Android Wear dominent actuellement le marché avec 61% des parts. La Motorola Moto 360 réalise actuellement une bonne partie des volumes vendus.

Son fonctionnement

Les montres sous Android Wear communiquent avec le téléphone uniquement via Bluetooth. Les applications Wearables sont incluses dans les applications Android. Par conséquent, pour utiliser une application Wearable (Wearable app) sur la montre connectée, il suffit de télécharger l'application Android (Handeled app) sur le téléphone depuis le Google Play Store et l'installer sur le téléphone. Il s'agit ensuite d'effectuer l'appairage Bluetooth entre la montre et le téléphone puis de suivre les instructions qui s'affichent sur la montre. Dès que l'appairage est effectué entre le téléphone et la montre, l'application Wearable sera automatiquement installée sur la montre. Cette dernière communique avec le téléphone via la couche Android « Data wearable layer » qui permet l'échange des petits message (de petite taille) pour préserver la batterie Fig.3.

Quels usages possibles ?

Plusieurs usages sont possibles avec les montres basées sur l'OS Android Wear.

Les notifications transmises par le téléphone vers la montre :

avoir un affichage spécifique de l'information sur un écran tactile qui permet d'y répondre rapidement.

Les applications autonomes installées sur la montre

les applications Android Wear sont des applications Android qui s'exécutent à 100 % sur la montre. L'avantage des applications sur l'Android Wear, c'est qu'on peut prendre avantage des informations des capteurs qui sont sur la montre et pas sur le téléphone (e.g., mesure de la fréquence cardiaque de l'utilisateur) Fig.4.

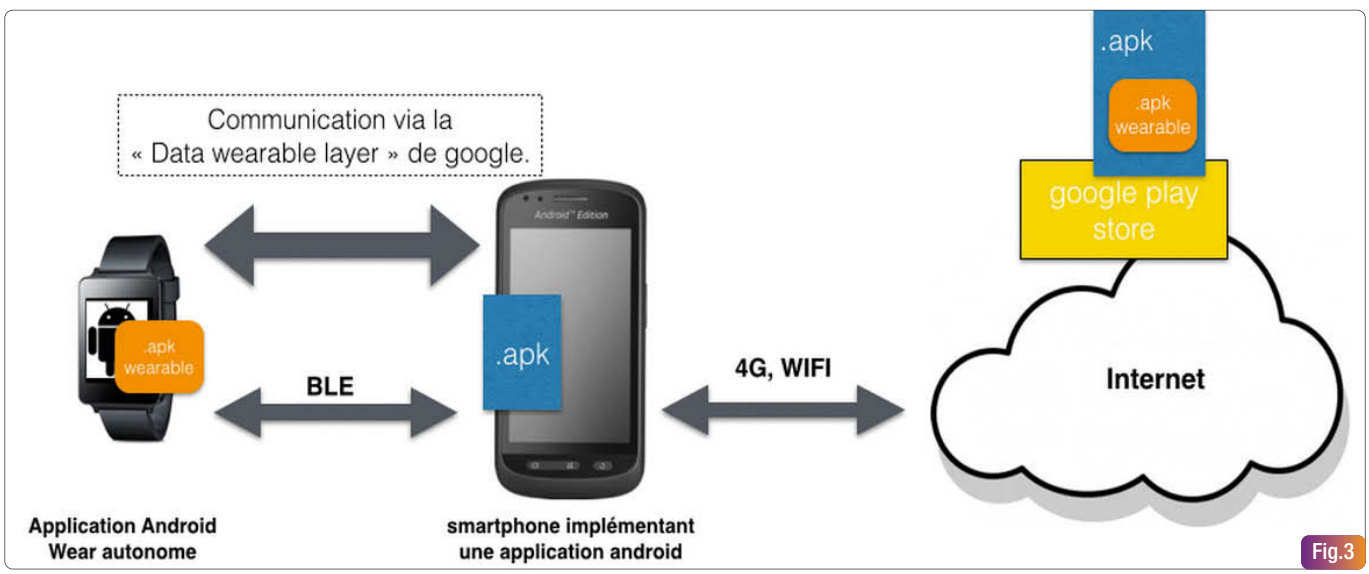


Fig.3

COMMENT DÉVELOPPER SA PREMIÈRE APPLICATION AVEC ANDROID WEAR?

Quelles sont les différences entre une application pour Wearable et une application pour téléphone ?

Dans cette partie, voyons ensemble comment développer une application Wearable permettant de mesurer le rythme cardiaque. Une “Wearable app” est une application s’exécutant directement sur la montre, ce qui permet d’avoir accès au hardware dont le moniteur cardiaque. Comme on l’a introduit dans la partie précédente, il existe quelques différences entre une Android app classique (pour téléphone) et une Wearable app :

- Un utilisateur final ne peut pas installer directement une application sur sa montre. En effet, le binaire de l’application Wear doit être embarqué dans une application Android (dite compagnon). À l’installation de cette application sur le téléphone, le binaire Wear app est automatiquement transféré et installé sur la montre. La suppression de l’application depuis le téléphone entraîne sa désinstallation automatique sur la montre. Néanmoins, il est possible d’installer directement une application sur sa montre via un câble usb lors de la phase de développement.
- Une Wearable app n’a accès qu’à un sous ensemble des APIs du SDK Android. Par exemple, les APIs android.webkit ou android.appwidget ne sont pas disponibles sur Wear. Impossible alors d’utiliser les webviews.
- La taille des applications Wearables doit rester la plus faible possible. Il faut déléguer au maximum le travail à l’application compagnon afin de n’embarquer que le strict nécessaire.
- Lorsqu’un utilisateur n’interagit pas avec ladite application, le système se met en veille pour préserver la batterie. Quand le système se réveille, le focus ne revient pas sur l’application mais sur la home de la montre. Pour afficher une information persistante, il vaut mieux utiliser le fil de notifications.

Développement de la Wear App

Il est temps de rentrer dans le vif du sujet et de développer une application de mesure du rythme cardiaque. Le plus simple est de créer un nouveau projet avec Android Studio en sélectionnant les options Phone and Tablet et Wear. Cliquez sur next puis sur finish jusqu’à la création du projet. Le projet se présente sous la forme de deux modules : l’un pour l’application téléphone (mobile) et le second pour l’application allant sur la montre (wear). En jetant un coup d’oeil au module wear, nous nous apercevons qu’il est effectivement très similaire à celui d’un projet Android classique. Les deux principaux changements sont les suivants : Dans le fichier AndroidManifest.xml, nous précisons que l’application s’exécute sur une montre

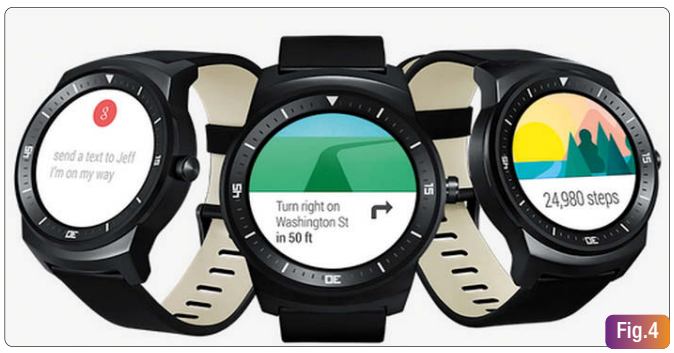


Fig.4

Exemples d’usages (de gauche à droite): envoi de messages, utilisation de GPS, comptage de nombre de pas.

<uses-feature android:name="android.hardware.type.watch" /> ;
Un nouveau composant fait son apparition dans le layout principal de l’application : le WatchViewStub. Il permet d’implanter un layout carré ou bien rond suivant les caractéristiques de la montre sur laquelle l’application s’exécute. Ce composant est fourni par la Wearable Support Library.

```
<android.support.wearable.view.WatchViewStub
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/watch_view_stub"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  app:rectLayout="@layout/rect_activity_main"
  app:roundLayout="@layout/round_activity_main"
  tools:context=".MainActivity"
  tools:deviceIds="wear">
</android.support.wearable.view.WatchViewStub>

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  final WatchViewStub stub = (WatchViewStub) findViewById(R.id.watch_view_stub);
  stub.setOnLayoutInflatedListener(new WatchViewStub.OnLayoutInflatedListener() {
  @Override
  public void onLayoutInflated(WatchViewStub stub) {
      heartRateTxtView = (TextView) stub.findViewById(R.id.text);
  }
  });
}
```

Côté module mobile, le principal changement est l’apparition d’une ligne dans le bloc dependencies :

```
wearApp project(':wear')
```

Cela indique que le module wear est une Wearable App qui doit être embarquée dans l’application Android (module mobile).
Pour pouvoir récupérer le rythme cardiaque d’un utilisateur, il faut tout d’abord ajouter la permission android.permission.BODY_SENSORS dans le fichier AndroidManifest.xml de l’application Weareable, mais aussi dans celui de l’application compagnon. En effet, il est impératif que toutes les permissions présentes dans le manifest de la Weareable App soient aussi présentes dans celui de l’application compagnon. Dans le cas contraire, l’application Weareable ne s’installera pas sur la montre. Cette contrainte permet de garantir que l’utilisateur aura connaissance de toutes les permissions requises par l’application (mobile + wearable).
Une fois la permission ajoutée, dans la méthode onCreate de la classe MainActivity, nous récupérons l’instance du SensorManager ainsi qu’une référence vers le moniteur cardiaque.

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
heartRateSensor = sensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
```

Dans la méthode onResume(), nous enregistrons un listener de type SensorEventListener si nous avons un moniteur cardiaque disponible. En effet, à l’heure actuelle certaines montres Android Wear comme la LG G

Watch, n'ont pas ce type de capteur. Cela nous donne le code suivant :

```
@Override
protected void onResume() {
  super.onResume();
  if (heartRateSensor == null) {
  heartRateTxtView.setText(R.string.no_sensor_available);
  } else {
  sensorManager.registerListener(this, heartRateSensor, SensorManager.SENSOR_DELAY
_NORMAL);
  }
}
```

La valeur SensorManager.SENSOR_DELAY_NORMAL indique à quels intervalles de temps nous souhaitons recevoir des mises à jour de la mesure. Les valeurs possibles sont : SENSOR_DELAY_NORMAL, SENSOR_DELAY_FASTEST, SENSOR_DELAY_GAME, SENSOR_DELAY_UI.

Il faut bien sûr penser à retirer le listener lors du onPause() :

```
@Override
protected void onPause() {
  super.onPause();
  if (heartRateSensor != null) {
  sensorManager.unregisterListener(this, heartRateSensor);
  }
}
```

Pour en revenir au SensorEventListener, il s'agit d'une interface dont les méthodes sont les suivantes :

- onSensorChanged(SensorEvent event) : appelé lorsque la mesure d'un capteur a changé ;
- onAccuracyChanged(Sensor sensor, int accuracy) : appelé lorsque la précision d'un capteur a changé.

Dans notre cas, nous nous attarderons surtout sur la première méthode, dont l'implémentation est la suivante :

```
@Override
public void onSensorChanged(SensorEvent event) {
  if (event.accuracy != SensorManager.SENSOR_STATUS_UNRELIABLE) {
  heartRateTxtView.setText(getString(R.string.heart_rate_frequency, event.values[0]));
  } else {
  heartRateTxtView.setText(getString(R.string.unreliable_heart_rate));
  }
}
```

Le SensorEvent est envoyé à chaque changement de valeur du capteur :
values : ensemble des valeurs de la mesure. Dans notre cas, le tableau ne contient qu'une seule entrée. Pour un accéléromètre, la taille du tableau est 3 (une mesure pour chaque axe x,y et z) ;
accuracy : précision de la mesure. Il s'agit d'une valeur discrète dont les valeurs possibles sont SENSOR_STATUS_LOW, SENSOR_STATUS_MEDIUM, SENSOR_STATUS_HIGH et SENSOR_STATUS_UNRELIABLE ;
sensor : sensor à l'origine de cet évènement ;
timestamp : moment de la mesure (temps en nanosecondes).
Au final en retouchant un peu les layouts, nous obtenons les écrans suivants : Fig.5 et 6.

Transmission des informations depuis la montre

Notre montre est maintenant capable de mesurer le rythme cardiaque mais ne transmet pas ces informations à l'application compagnon. Ainsi, pour pouvoir communiquer entre la montre et le téléphone, utilisons la Data Layer API. Cette API permet de synchroniser des informations entre les appareils. Afin d'y accéder, nous allons utiliser les Play Services.
Nous devons instancier un objet GoogleApiClient dans le onCreate de notre MainActivity sur le projet Wear, se connecter dans le onStart() et se déconnecter dans le onStop().

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  [...]
  googleApiClient = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(new GoogleApiClient.ConnectionCallbacks() {
      @Override
      public void onConnected(Bundle connectionHint) {
      }

      @Override
      public void onConnectionSuspended(int cause) {
      }
    })
    .addOnConnectionFailedListener(new GoogleApiClient.OnConnectionFailedListener() {
      @Override
      public void onConnectionFailed(ConnectionResult result) {
      }
    })
    .addApi(Wearable.API)
    .build();
}

@Override
protected void onStart() {
  super.onStart();
  googleApiClient.connect();
}

@Override
protected void onStop() {
  googleApiClient.disconnect();
  super.onStop();
}
```

Maintenant que nous sommes connectés, nous allons pouvoir transmettre nos informations. Il existe deux manières principales d'envoyer des données entre appareils :

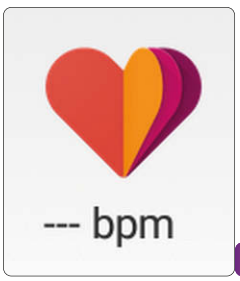
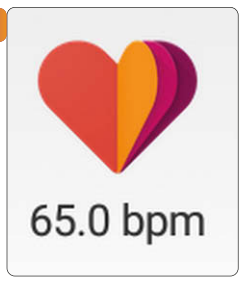


Fig.5

Fig.6


- Soit on envoie un DataItem : un objet de données synchronisées entre les deux appareils (quand il est modifié sur un appareil, l'autre appareil est notifié automatiquement) ;
- Soit on envoie un Message : une communication à voie unique avec des données non synchronisées, du type « fire-and-forget » : on envoie un signal et on ne se soucie pas du reste.

Dans notre cas, nous allons privilégier la première option en transmettant nos informations à l'aide d'un DataItem. Ce dernier est en quelque sorte une map (bundle) qui sera marshallée pour être envoyée par bluethooth au téléphone. Notre méthode onSensorChanged devient alors :

```
@Override
public void onSensorChanged(SensorEvent event) {
  [...]
  if (googleApiClient.isConnected()) {
  PutDataMapRequest dataMapRequest = PutDataMapRequest.create("/heartrate");
  DataMap dataMap = dataMapRequest.getDataMap();
  dataMap.putFloat("HeartRate", event.values[0]);
  Wearable.DataApi.putDataItem(googleApiClient, dataMapRequest.asPutDataRequest());
  } else {
  [...]
  }
}
```

La valeur "/heartrate" correspond au path associé à cette requête. Ce path est similaire au chemin d'une ressource dans une API REST. Pour finir, Il faut bien penser à vérifier que le client est connecté avant de faire un appel à la DataApi.

Réception des informations depuis le téléphone

A partir de maintenant, la montre synchronise à intervalle régulier la valeur du rythme cardiaque à l'aide de la DataApi. Deux possibilités s'offrent à nous pour pouvoir récupérer cette information :

- Soit depuis un service héritant de WearableListenerService ;
- Soit depuis une activité implémentant DataApi.DataListener.

Comme nous souhaitons simplement afficher l'information sur une activité du téléphone, nous privilégions la seconde option.
Pour recevoir l'information dans notre activité, nous avons, à nouveau, besoin d'un GoogleApiClient (cf 5.3). Dans le callback onConnected(), nous enregistrons un listener pour écouter les évènements de la DataApi :

```
@Override
public void onConnected(Bundle connectionHint) {
  Wearable.DataApi.addListener(googleApiClient, MainActivity.this);
}
```

Listener qu'il faut retirer lors du onStop() :

```
@Override
protected void onStop() {
  if (googleApiClient != null && googleApiClient.isConnected()) {
  Wearable.DataApi.removeListener(googleApiClient, this);
  googleApiClient.disconnect();
  }
  super.onStop();
}
```

A présent, nous serons notifiés dès qu'un item de la couche de données sera modifié. L'interface DataApi.DataListener expose une méthode onDataChanged(DataEventBuffer dataEvents).

```
@Override
public void onDataChanged(DataEventBuffer dataEvents) {
  for (DataEvent event : dataEvents) {
  Uri uri = event.getDataItem().getUri();
  if (event.getType() == DataEvent.TYPE_DELETED) {
        Log.d(TAG, "DataItem deleted: " + uri);
  } else if (event.getType() == DataEvent.TYPE_CHANGED) {
        Log.d(TAG, "DataItem changed: " + uri);
  if("/heartrate".equals(uri.getPath())){
    DataMapItem dataMapItem =  DataMapItem.fromDataItem(event.getDataItem());
          Float heartRate = dataMapItem.getDataMap().getFloat("HeartRate");
          [...]
        }
}
  }
}
```

Nous constatons qu'un DataEvent peut être de deux types : TYPE_DELETED ou TYPE_CHANGED. La valeur TYPE_CHANGED indique qu'un item a été ajouté ou bien modifié. A partir de là, nous vérifions que le path de l'URL correspond bien à "/heartrate". Nous pouvons ensuite extraire la valeur du rythme cardiaque à l'aide de la classe DataMapItem.

CONCLUSION

Au travers de cet article, nous nous sommes focalisés sur la technologie des montres connectées basée sur l'OS Android Wear. De plus, l'année 2015 s'annonce comme étant l'année des montres connectées en particulier avec la sortie l'Apple Watch qui risque de concurrencer la domination des montres Android Wear.

Xebia

Xebia est un cabinet de conseil parisien spécialisé dans les technologies Big Data, Cloud et Web, les architectures Java et mobilité dans des environnements agiles. Nous croyons qu'il est possible de développer des applications à la fois innovantes, performantes et de très bonne qualité, et c'est au sein de Xebia Studio que nous mettons en œuvre cette philosophie. Xebia fait aujourd'hui autorité dans le domaine mobile et souhaite continuer sur cette lancée avec l'avènement des objets connectés. Pour ce faire, les xebians partagent leurs connaissances au quotidien via notre blog technique (blog.xebia.fr), twitter (@XebiaFr) et participent/animent des conférences et des Techevents.

Accéder à SharePoint 2013 via l'API REST en Objective-C

Tout au long de ces dernières années, SharePoint est devenue une plateforme majeure dans différents domaines au sein des entreprises, allant de la collaboration à la Business Intelligence en passant par la recherche ou les réseaux sociaux d'entreprise.

Stéphane Cordonnier

En parallèle, la démocratisation des périphériques mobiles (notamment les smartphones et tablettes) a vu le jour au sein de ces mêmes entreprises, et, logiquement les utilisateurs souhaitent pouvoir retrouver sur ces périphériques leurs données stockées dans SharePoint .
Depuis quelques mois, Microsoft propose plusieurs applications permettant de répondre à ces besoins. Si l'on se concentre sur l'iPhone et l'iPad, l'arrivée d'Office sur les plateformes mobiles Apple, conjuguée avec l'existence d'autres applications telles que OneDrive, Newsfeed, Outlook, Lync ou encore Dynamics CRM, permet de proposer des outils professionnels connus de la plupart des utilisateurs.
Toutefois, il arrive que certaines entreprises aient besoin d'aller plus loin que le simple fait d'utiliser les applications proposées par Microsoft, en créant leurs propres applications interconnectées avec le système d'information de l'entreprise, dont SharePoint peut représenter une brique majeure. Mais une image tenace subsiste dans l'esprit de beaucoup de personnes : les mondes Apple et Microsoft ont du mal à coexister. Pourtant Microsoft a entrepris depuis pas mal de temps maintenant, et cela va continuer dans les années à venir, une démarche d'ouverture en permettant à des solutions autres que celles développées en .NET, de pouvoir s'interconnecter avec l'écosystème proposé par la firme de Redmond. SharePoint ne déroge pas à cette règle et possède un ensemble de services REST permettant à n'importe quelle application écrite en Objective-C, Java, PHP... de pouvoir s'interfacer et manipuler les données d'entreprise stockées au sein de cette plateforme.

Présentation de l'API REST de SharePoint 2013

Depuis sa version 2003, SharePoint propose un ensemble de services Web (ASMX / SOAP) qu'il était relativement compliqué d'utiliser dans un monde externe à Microsoft et à sa technologie .NET.
SharePoint 2010 a corrigé en partie cela en proposant des services basés sur WCF (client.svc) et qui permettaient de pouvoir simplifier l'intégration avec d'autres systèmes, mais ils étaient limités fonctionnellement et la construction des URLs n'étaient pas forcément très simple.
Avec SharePoint 2013, Microsoft a revu et enrichi de manière importante son système de services Web (toujours basés sur WCF) afin de proposer un système en adéquation avec les standards actuels et notamment l'utilisation de REST pour la construction et l'appel des URLs.
Comme souvent avec SharePoint, tout part de l'URL d'un site (ex : https://monsite.sharepoint.com) à laquelle il est possible de concaténer différentes sections (endpoints techniquement parlant) permettant de récupérer des informations.
Ci-dessous, quelques exemples des « endpoints » les plus utilisés :

- https://monsite.sharepoint.com/_api/Site → Informations générales sur la collection de sites courante
- https://monsite.sharepoint.com/_api/Web → Informations générales sur le site courant
- https://monsite.sharepoint.com/_api/Lists → Accès aux listes / bibliothèques du site courant
- https://monsite.sharepoint.com/_api/Search → Accès au moteur de recherche

Comme on peut le voir, pour construire une URL permettant d'accéder aux données stockées dans SharePoint, il suffit d'ajouter la valeur « _api » derrière l'URL du site, puis d'ajouter ensuite l'élément auquel on souhaite accéder.
Cette convention est également valable si jamais un site possède des sous-sites et que l'on souhaite manipuler les données stockées dans ces sous-sites. Par exemple, pour accéder aux informations générales d'un sous-site, l'URL devant être appelée serait de la forme « https://monsite.sharepoint.com/subsite/_api/Web ».
Les données renvoyées lorsque l'on tente d'accéder à une URL de cette forme sont par défaut au format XML, mais elles peuvent être renvoyées par le serveur au format JSON en le spécifiant dans les en-têtes HTTP de la requête soumise au serveur, comme nous le verrons ultérieurement.
Vous pouvez voir un exemple dans la capture d'écran ci-dessous Fig.1.

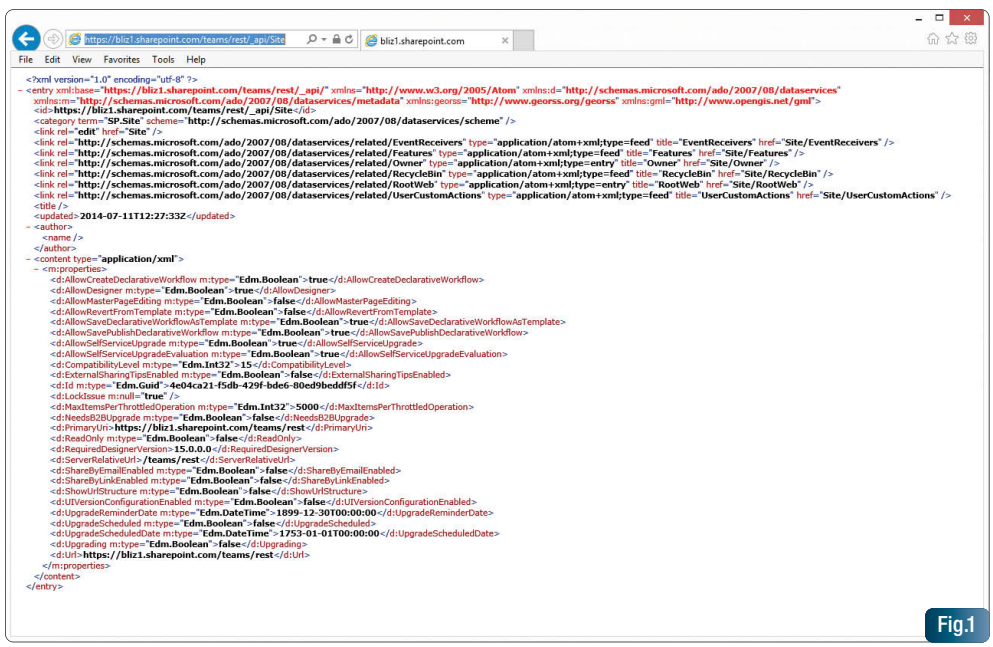


Fig.1

Interrogation d'une liste / bibliothèque de documents

Maintenant que nous avons évoqué les grands principes sur la manière dont fonctionnent les services REST offerts nativement avec SharePoint 2013, voyons comment accéder à ces informations depuis une application écrite en Objective-C (qu'elle soit destinée à MacOS, iPhone ou iPad).

La première étape consiste à définir quelles classes disponibles dans le Framework Foundation seront utilisées pour créer une couche de communication réseau. Pour cela nous avons deux possibilités offertes à savoir utiliser NSURLConnection (disponible sur toutes les plateformes) ou bien NSURLSession (disponible uniquement à partir de Mavericks et iOS 7).

Afin de couvrir le maximum de plateformes possibles, nous utiliserons ici NSURLConnection, mais il est bon de noter qu'Apple recommande désormais d'utiliser NSURLSession qui apporte une plus grande simplicité d'utilisation et quelques ajouts fonctionnels.

```
NSURL *url = [NSURL URLWithString:@" https://monsite.sharepoint.com/_api/Site"];
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:requestURL
 cachePolicy:NSURLRequestReloadIgnoringLocalAndRemoteCacheData timeoutInterval:30];
NSURLConnection *connection = [[NSURLConnection alloc] initWithRequest:request delegate
:self startImmediately:NO];
[connection scheduleInRunLoop:[NSRunLoop mainRunLoop] forMode:NSRunLoopCommon
Modes];
[connection start];
```

Le code ci-dessus permet de déclarer l'URL du site auquel nous souhaitons nous connecter, puis de créer une requête associée à cette URL précisant que nous souhaitons systématiquement interroger le serveur en ignorant la gestion du cache, et en définissant un délai maximum de 30s pour exécuter cette requête.

Nous créons ensuite une connexion basée sur cette requête en précisant qu'elle ne doit pas démarrer immédiatement après sa création. De plus la classe dans laquelle nous déclarons ce code servira de délégué pour gérer les évènements associés (réception d'une réponse, demande d'authentification, réception de données, déclenchement d'une erreur...) suivant le pattern habituel au sein d'API mises à disposition par Apple.

Nous indiquons ensuite que cette connexion sera exécutée au sein de la boucle d'exécution principale de l'application, puis nous terminons par le démarrage de la connexion qui aura pour but d'initier le flux de communication réseau avec le serveur.

Afin de gérer tous les évènements en rapport avec notre connexion, nous avons besoin de définir les méthodes de notre délégué qui doivent être interceptées. Pour cela, il est nécessaire au préalable de définir que notre classe courante est capable d'intercepter ces évènements en implémentant le protocole NSURLConnectionDataDelegate comme suit.

```
@interface SharePointWebRequest : NSOperation <NSURLConnectionDataDelegate>
…
@end
```

Une fois cette déclaration effectuée, nous pouvons utiliser les méthodes qui nous intéressent afin de pouvoir intercepter les évènements utiles.

```
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse
*)response
{
   _buffer = [NSMutableData data];
}
- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationChallenge:
(NSURLAuthenticationChallenge *)challenge
{
   _challengeAuthenticationCount++;
   if (_challengeAuthenticationCount > 2) {
      [[challenge sender] cancelAuthenticationChallenge:challenge];
      return;
   }
   NSURLCredential *credentials = [NSURLCredential credentialWithUser:_username password:
_password persistence:NSURLCredentialPersistenceNone];
   [[challenge sender] useCredential:credentials forAuthenticationChallenge:challenge];
}
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data
{
   [_buffer appendData:data];
}
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error
{
   // Erreur lors du téléchargement
}
- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
   // Données récupérées et prêtes à être utilisées
}
```

Comme nous le voyons, de nombreux évènements nous sont utiles afin de gérer le cycle de vie d'une requête Web. Le premier d'entre eux (didReceiveResponse) permet d'instancier dans une variable privée (nommée _buffer) un objet de type NSMutableData qui permettra de stocker les données binaires reçues. La notion de « Mutable », présente dans le nom de la classe, permet de définir que nous pourrons ajouter des données à ce buffer au fur et à mesure de leur réception (contrairement à NSData qui correspond à un type auquel il est impossible d'ajouter des données).

L'évènement suivant (didReceiveAuthenticationChallenge) permet de gérer les demandes d'authentification qui pourraient être émises par le serveur. Ici nous prenons un exemple simple où SharePoint fonctionnerait sur un mode d'authentification basé sur un compte Windows représenté par un couple user/password. Pour cela nous construisons un objet NSURLCredential basé sur ces données, que nous fournissons ensuite au serveur. Si l'authentification échoue 3 fois, nous annulons la demande d'authentification.

Vient ensuite le tour de l'évènement de réception de données (didReceiveData) dans lequel nous nous contentons d'ajouter les données reçues (et passées en paramètre de la méthode) au buffer créé précédemment. Enfin les deux derniers évènements nous permettent de savoir si une erreur a eu lieu durant le téléchargement (didFailWithError) ou bien si le téléchargement s'est correctement déroulé (connectionDidFinishLoading), et nous permet de pouvoir traiter les données reçues.

Utilisation des données téléchargées

Comme nous l'avons évoqué précédemment, par défaut les données téléchargées depuis les services REST sont renvoyées au format XML. Si vous préférez que ces données soient renvoyées au format JSON, pour des raisons de performance/volume de données notamment, il est possible de demander à SharePoint de formater celles-ci comme vous le souhaitez.

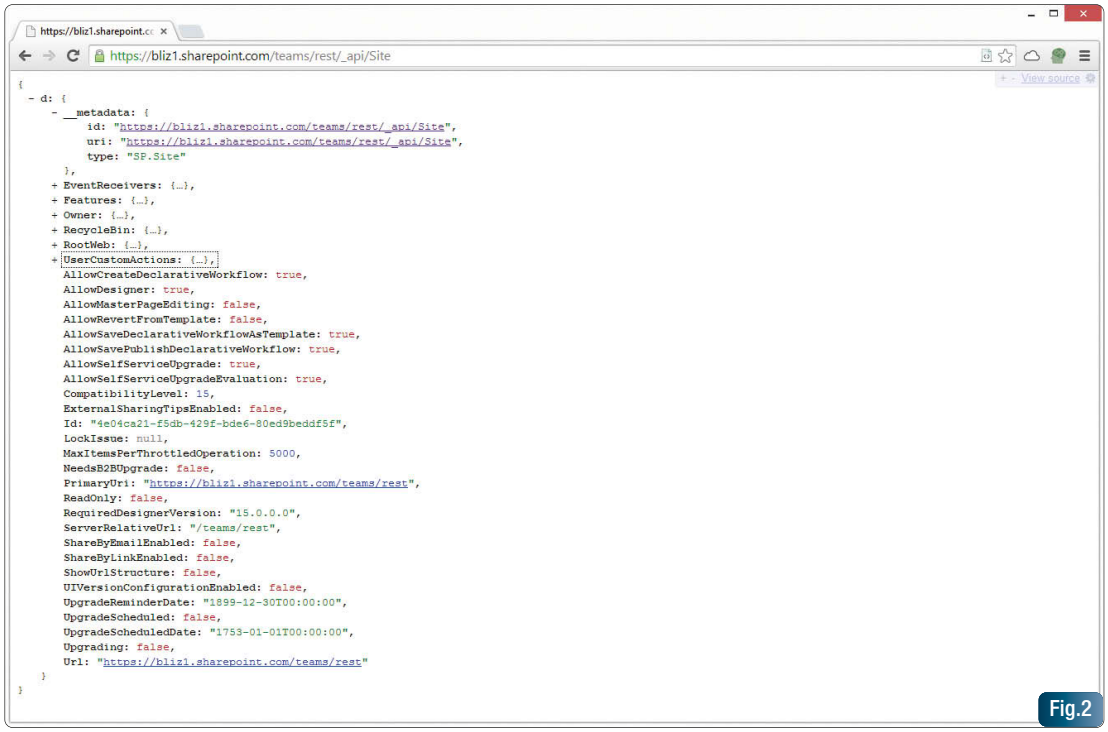


Fig.2

Pour cela, il suffit de le préciser dans l'en-tête HTTP « Accept » de la rêquete comme vous pouvez le voir ci-dessous :

```
[request setValue:@"application/json;odata=verbose" forHTTPHeaderField:@"Accept"];
```

Une fois la requête modifiée comme évoquée ci-dessus, à chaque fois que vous exécuterez celle-ci, les résultats retournés par le serveur seront formatés en JSON. Vous pouvez voir un exemple de cette requête au format JSON ci-dessus Fig.2.
Le fait que les données soient renvoyées directement au format JSON va nous faciliter la vie en vue de leur exploitation ; car nous avons tout ce qu'il faut dans Foundation pour traiter ce type de données avec notre code Objective-C.
En effet, dans les classes mises à disposition par Apple, il en existe une nommée NSJSONSerialization qui permet comme son nom l'indique, de pouvoir sérialiser/dé-sérialiser des objets au format JSON.

```
NSError *error = nil;
id json = [NSJSONSerialization JSONObjectWithData:_buffer options:NSJSONReadingMutable
Containers error:&error];
if (error) {
   // Erreur de la lecture du flux JSON
} else {
   NSDictionary *root = [json objectForKey:@"d"];
   NSString *url = [root objectForKey:@"Url"];
}
```

Dans le code ci-dessus, nous commençons par déclarer une variable destinée à recevoir l'erreur éventuelle qui pourrait être générée si les données renvoyées par le serveur ne pouvaient être lues, par exemple, si elles sont mal formatées.
La ligne suivante est le cœur du système car, elle permet, à partir des données stockées dans notre buffer au format binaire (dans un objet NSData), de pouvoir les transformer en objets utilisables directement depuis Objective-C (NSDictionary, NSArray, NSString, NSNumber...).
Pour finir et si la lecture des données au format JSON n'a pas généré d'erreurs, nous sommes désormais à même de pouvoir les exploiter.
Si vous regardez attentivement la capture d'écran précédente, le flux JSON renvoyé par le serveur est composé d'un premier dictionnaire de données (marqué par une accolade suivant la convention habituelle) qui contient une unique clé nommée « d » et dont la valeur est un autre dictionnaire, contenant cette fois toutes les informations de notre site. Notre code permet donc de récupérer dans le flux JSON, l'objet qui possède la clé nommée « d » puis de le stocker dans une variable typée en tant que dictionnaire (NSDictionary). A partir de ce dictionnaire, nous récupérons ensuite la valeur de la clé « Url » que nous stockons dans une variable de type NSString. A noter que les noms de clés dans le dictionnaire sont sensibles à la casse.
Si nous souhaitons récupérer d'autres valeurs dans le dictionnaire, il suffit d'écrire des lignes similaires à la dernière de notre exemple de code précédent, en changeant la valeur de la clé qui nous intéresse (ex : Id, PrimaryUri, ServerRelativeUrl...).

Conclusion

Comme nous l'avons vu dans cet article, SharePoint 2013 permet désormais, via les services REST mis à disposition nativement, de pouvoir s'interfacer avec n'importe quelle plateforme capable de soumettre des requêtes HTTP vers le serveur.
Bien évidemment, les cas d'utilisation dont vous aurez besoin quotidiennement iront bien plus loin que le fait de simplement lire les informations d'un site (ex : requêter une liste, mettre à jour des fichiers, effectuer des recherches...) mais le principe est globalement similaire. La plus grosse différence résidera dans le « endpoint » qui sera utilisé ou bien dans le type de requête que vous enverrez au serveur (GET, POST, PUT, DELETE...).
Si vous souhaitez plus d'informations sur toutes les possibilités offertes par SharePoint 2013 et ses services REST, Microsoft a mis à disposition sur son site MSDN de nombreux articles décrivant toutes les fonctionnalités de manière claire. Reste pour vous à transcrire les informations en code Objective-C afin de transmettre les bonnes requêtes au serveur et traiter les flux JSON renvoyés en conséquence.
Plus d'informations sur http://msdn.microsoft.com/en-us/library/office/fp142380(v=office.15).aspx

Big Brother is Watching You… ou l'inverse

Qui n'a jamais rêvé d'un ordinateur qui lui réponde au doigt… et à l'œil ? C'est désormais possible. Plus besoin de gesticuler devant son écran avec Kinect, plus besoin non plus de déplacer sa souris, ni même de taper au clavier. Tout ce que vous aurez à faire désormais c'est de regarder votre écran.

Michaël Fery et Maxime Frappat
Consultants
Infinite Square

Laissez-moi m'expliquer un peu en vous présentant un produit fun et abordable répondant au doux nom de « Tobii EyeX Controller ». Fun parce que cet appareil au look de Kinect va vous permettre de traquer votre regard afin d'interagir avec des applications. Ajoutons qu'il est abordable car le kit de développement est à 99€ seulement.

Faisons connaissance avec la bête **Fig.1**.

Fig.1

Dès le déballage de la boîte on remarque que l'appareil est fourni avec un et un seul câble en USB 3 pour son alimentation et la communication avec votre ordinateur. Pas besoin de sortir la multiprise, ouf. La deuxième bonne idée est l'ensemble de bandes magnétiques autocollantes que l'on peut disposer au-dessus ou en dessous de son écran et qui est destiné à maintenir par aimantation le traqueur. Une fois les drivers installés et l'appareil branché, nous sommes invités à effectuer un calibrage durant lequel nous devons regarder un point lumineux se déplacer sur notre écran. C'est ainsi que l'appareil apprend où nos yeux se portent à tout moment lorsque nous regardons notre écran. Le calibrage se fait en 30 secondes montre en main ! Nous pouvons alors profiter directement des fonctionnalités offertes par ce nouveau device. Les possibilités sont multiples : nous allons pouvoir sélectionner des éléments, se déplacer dans le menu démarrer Windows 8 ou scroller dans notre navigateur préféré (via des plugins). Nous pouvons même jouer du ALT+Tab de manière visuelle ou téléporter notre souris là où notre regard se pose sur l'écran (attention au mal de mer). Tout cela est très intéressant mais si nous avons un kit de développement, ce n'est pas pour rien, alors mettons un peu les mains dans le cambouis et développons quelque chose.

Le SDK .NET

La brique de base dont nous allons avoir besoin pour créer nos applications intégrant le traqueur est le kit de développement. En parcourant la partie du site de Tobii destinée aux développeurs nous pouvons voir qu'il existe 3 sortes de SDK pour le moment : .NET, C/C++ et Unity. Nous allons télécharger le SDK qui nous convient et pour commencer, nous allons choisir le SDK .NET qui nous permettra d'utiliser le langage C# que nous apprécions tout particulièrement. Le SDK .NET prend pour le moment en charge les projets WinForms et WPF mais pas encore les applications Windows Store à cause de limitations techniques **Fig.2**.

Données brutes

A la manière du Kinect, il existe plusieurs types d'interactions avec le SDK : soit en récupérant les données quasi-brutes de l'appareil que nous pourrons traiter, soit en utilisant les composants intégrant déjà une logique d'interaction visuelle prémâchée pour nous.

La première des possibilités offertes par le SDK est par exemple de récupérer les données du capteur, nous indiquant l'emplacement exact au pixel près où le regard de l'utilisateur est porté sur l'écran **Fig.3**.

Nous pouvons également connaître la position 3D de chaque œil par rapport au capteur **Fig.4**.

Cela vous permet de créer votre propre logique, et de recréer votre propre surcouche au SDK existant, un peu de la même manière que vous pourriez interpréter les mouvements de la souris au lieu de vous abonner aux événements « classiques » de Windows (click gauche, click droit, etc.).

Ce que nous allons cependant voir plus en détails ici est l'utilisation des composants existants dans une application WPF.

Préparation de la solution et utilisation de la dll

Pour commencer, créons une solution nommée « EyeTracker » dans laquelle nous ajoutons un projet WPF que nous appellerons « EyeTracker.WPF » .

Il est ensuite nécessaire de référencer le SDK EyeX DotNet. Pour cela, 2 possibilités existent. La première est de profiter du fait que les sources du SDK sont également disponibles pour intégrer le projet. Cette solution sera utile si vous souhaitez modifier les comportements définis dans le SDK et sera donc réservée aux développeurs plus avertis. La seconde est d'utiliser les librairies .NET déjà compilées comme nous le ferons dans l'exemple qui suit.

Attention, si vous essayez d'utiliser ces sources en l'état, le programme risque de vous dire qu'il n'arrive pas à trouver la dll Tobii.EyeX.Client.dll. Le SDK étant un wrapper autour de cette dll, il est impératif de l'embarquer dans notre solution. Heureusement celle-ci est également fournie dans le dossier lib du SDK. Il vous reste donc à l'ajouter à votre projet EyeTracker.WPF afin qu'elle soit embarquée et disponible avec votre application. Attention aussi à bien mettre son contenu à « Copy to Output Directory ».

Fig.2

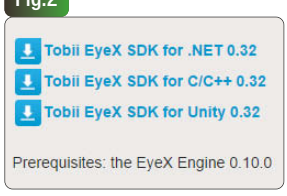


Fig.4

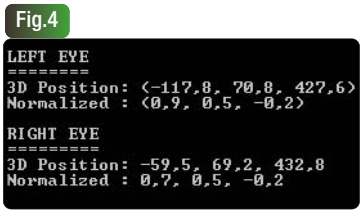


Fig.3

Créer un composant réactif à notre regard

Nous avons désormais accès aux fonctionnalités du SDK EyeX et nous allons donc les intégrer à notre page WPF. En bons développeurs, nous sommes fainéants et nous allons donc créer un style dont hériteront les éléments que nous souhaiterons être activables sans avoir à les reconfigurer chacun un à un. Commençons par définir le namespace xml :

```
xmlns:eyeX="clr-namespace:EyeXFramework.Wpf;assembly=EyeXFramework"
```

Puis nous pouvons créer notre style :

```
<style TargetType="FrameworkElement" x:key="EyeXAwareElement">
    <setter Property="eyeX:Behavior.GazeAware" Value="True" />
    <setter Property="eyeX:Behavior.GazeAwareDelay" Value="250" />
</style>
```

Ce style définit la propriété GazeAware à true pour indiquer au SDK que l'élément est sensible au regard. La seconde, GazeAwareDelay, permet de définir un délai avant lequel le bouton est informé du regard qu'on pose sur lui. Selon l'application que vous souhaitez développer, cette valeur en millisecondes sera à ajuster afin d'éviter d'avoir une IHM qui clignote comme un sapin de Noël. Dans notre page nous créons alors un bouton pour tester le comportement en lui appliquant ce style :

```
<Button Margin="170" Content="Regarde-moi droit dans les yeux" Style="{StaticResource EyeXAwareElement}"/>
```

Désormais ce bouton est « aware ». Ce qui veut dire qu'il est conscient et potentiellement réactif au regard que l'on porte sur lui. Nous avons choisi un bouton, mais nous pouvons appliquer ce style à n'importe quel FrameworkElement si nécessaire.
Afin de rendre ce comportement plus visible, nous allons ajouter des triggers de comportement à ce bouton dès que le regard se porte sur lui. Pour cela nous ajoutons un style héritant du premier, qui sera utile pour nos boutons, et nous modifions donc le style de notre bouton actuel :

```
<style x:key="EyeXAwareButton" BasedOn="{StaticResource EyeXAwareElement}" TargetType="">
  <setter property="Background" value="LightSteelBlue" />
  <style.triggers>
    <trigger Property="eyeX:Behavior.HasGaze" Value="True">
      <setter Property="Background" Value="Red" />
    </trigger>
  </style.triggers>
</style>

<Button Margin="170" Content="Regarde-moi droit dans les yeux" Style="{StaticResource EyeXAwareButton}"/>
```

Avec ce code, le bouton est aware, et il passera rouge dès que nous porterons notre regard sur lui. En quelques lignes de code, nous venons de créer un bouton... timide ! Ce comportement ne pourra cependant avoir lieu que si nous donnons à notre application l'autorisation d'utiliser le capteur.

Démarrage du EyeXHost

Pour indiquer à notre application qu'elle doit utiliser le service de tracking, il faut qu'elle instancie un objet WpfEyeXHost, et le démarre. Dans notre cas, nous conservons une instance globale à l'application, mais il serait possible de ne le faire que sur un écran particulier si cela était pertinent.

Aussi, comme le SDK utilise des ressources non managées (rappelez-vous l'utilisation de la DLL à inclure au projet), il est impératif d'appeler la méthode Dispose pour les libérer lorsque l'application est quittée.

```

public partial class App : Application
{
        private readonly WpfEyeXHost _eyeXHost;
        public App()
        {
            _eyeXHost = new WpfEyeXHost();
            _eyeXHost.Start();
        }
        protected override void OnExit(ExitEventArgs e)
        {
            _eyeXHost.Dispose();
            base.OnExit(e);
        }
}

```

Voilà, nous venons de créer notre première application contenant un bouton réactif à notre regard.

Si vous souhaitez aller plus loin, sachez que vous pourriez faire réagir votre application aux évènements fournis par la classe WpfEyeXHost tels que « EyeTrackingDeviceStatusChanged » ou « UserPresenceChanged », par exemple. Je vous invite donc à fouiner pour découvrir les autres fonctionnalités offertes par le SDK.

SDK Unity

Attaquons-nous maintenant à une autre possibilité : utiliser le SDK Unity3D afin de découvrir comment nous pouvons intégrer le capteur Tobii EyeX à nos jeux vidéo.
De la même manière qu'en WPF, nous allons apprendre à rendre nos objets visuels 3D réactifs à notre regard.
Dans cet exemple nous créerons une sphère avec un effet de particules qui s'animera lorsque nous la regarderons.

Création du projet Unity

La première étape consiste bien évidemment à installer Unity3D. Pour rappel, toutes les informations nécessaires sont disponibles sur le site Unity3d : http://unity3d.com. Aucune contrainte de version n'est explicitée sur le site de Tobii mais il fonctionne parfaitement avec les dernières versions de Unity, alors téléchargez la dernière sans crainte !
Dans Unity, créez un nouveau projet « File », « New Project » puis dans la fenêtre d'assistant entrez le nom EyeTracker.Unity pour votre projet et enfin, cliquez sur « Create » **Fig.5**.

Installation du SDK

Sur le site de Tobii, ouvrez la page de téléchargement située dans la partie destinée aux développeurs : http://developer.tobii.com/downloads/
Sur cette page vous pourrez trouver et télécharger le SDK spécifique à Unity **Fig.6**.
Une fois le package téléchargé, vous pourrez décompresser le dossier et double-cliquer sur le fichier **EyeXFramework.unityPackage**. Une fenêtre s'ouvrira alors dans Unity vous proposant d'importer le package **Fig.7**.
Vous n'aurez alors qu'à cliquer sur « Import » pour que les assets soient importés dans votre projet courant **Fig.8**.

Ajout de la sphère à animer

Sélectionnez « GameObject », « CreateOther », « Sphere ».
Sur cette sphère nous allons rajouter deux composants. L'un pour rendre la sphère animée avec un effet de particules. L'autre pour rendre cette animation activable lorsque le regard se pose sur la sphère.
Sélectionnez donc la sphère et dans la vue « Inspector », sélectionnez « Add Component » puis dans le sous-menu « Effects », choisissez « Particle System ». Des particules partent désormais de la sphère.
Nous allons conditionner l'affichage de ces particules. Nous pouvons les désactiver par défaut en décochant la propriété Emission du composant « Particle System ». De plus, pour avoir un rendu plus réactif des changements d'état, je vous invite également à modifier la propriété « Start Lifetime » dans « Particle System », « Sphere » à 1 afin que les particules meurent plus rapidement lorsque notre regard se détournera.

Sphère interactive

Sélectionnez à nouveau la sphère, sélectionnez « Add Component » et dans le sous menu « Tobii EyeX », choisissez « Gaze Aware ». Nous avons ainsi créé un composant de type « GazeAwareComponent ». Ces scripts sont disponibles car vous avez importé précédemment le SDK Tobii dans Unity. La sphère est désormais sensible au regard qu'on lui porte. Lorsque nous la regardons, la propriété « HasGaze » de ce composant sera à true. Ajoutons alors un script à notre sphère pour lui donner la capacité de savoir quand nous la regardons, et donc d'aviser comment réagir en conséquence.
Pour cela, on sélectionne « Add Component », « New Script » puis « Create and Add ». Double cliquez sur ce script dans la fenêtre « Inspector » et l'éditeur de script C# configuré dans Unity s'ouvrira.
Il faudra alors mettre à jour le code qu'il contient de cette manière :

```
using UnityEngine;
using System.Collections;

public class ParticleBehavior : MonoBehaviour
{
    private ParticleSystem _particleComponent;
    private GazeAwareComponent _gazeAwareComponent;

    // Use this for initialization
    void Start ()
    {
        _particleComponent = GetComponent<ParticleSystem>();
        _gazeAwareComponent = GetComponent<GazeAwareComponent>();
    }

    // Update is called once per frame
    void Update ()
    {
        if (_gazeAwareComponent.HasGaze)
        {
            _particleComponent.enableEmission = true;
        }
        else
        {
            _particleComponent.enableEmission = false;
        }
    }
}
```

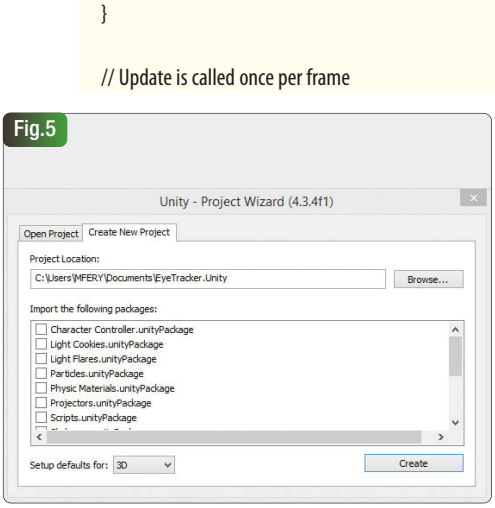
À chaque boucle de rendu (dans le corps de la méthode Update donc), un test sera effectué pour vérifier si la sphère est sous le regard de l'utilisateur. Si tel est le cas, alors l'animation de particules sera activée, sinon celle-ci sera désactivée.

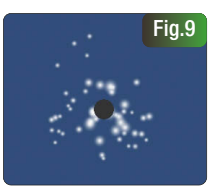
Testons le rendu

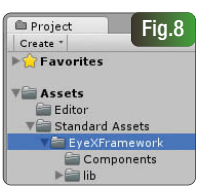
Maintenant que nous avons mis en place toute notre logique, il ne nous reste qu'à vérifier si le comportement attendu fonctionne. Pour cela, cliquez sur le bouton « Play » et vous pourrez observer votre sphère s'illuminer lorsque vous la regarderez, comme par magie **Fig.9**.
Il s'agit ici d'un scénario assez simple mais imaginez un jeu de suspense où cette technologie serait utilisée pour animer des personnages uniquement si vous ne les regardez pas : cela deviendrait vite très inquiétant et vous seriez vite immergé dans le jeu !

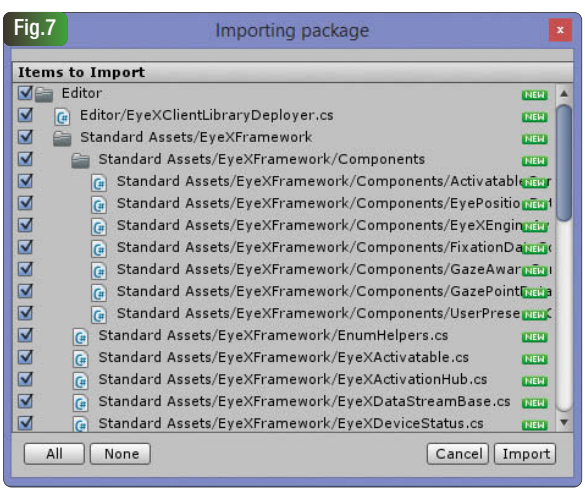
Conclusion

Nous avons vu dans cet article comment il peut être facile d'intégrer le capteur Tobii à nos applications, qu'elles soient en WPF ou Unity. Il est également à noter que depuis la dernière version, le SDK fournit également la possibilité d'effectuer la calibration depuis n'importe quelle application. Cette fonction pourra s'avérer utile si l'application est destinée à accueillir plusieurs utilisateurs.
Le fabricant du capteur fournit également des plugins en C/C++ sur son site ou un plugin pour Unreal Engine 4 directement sur GitHub. Certains SDK bas niveau sont également disponibles pour Linux et Android. De quoi s'amuser sans limite !
Concernant les prochaines versions, aucune annonce n'est faite mais les mises à jour sont nombreuses et régulières alors attendons-nous à découvrir de nouvelles fonctionnalités prochainement.









JavaScript pour les Jedi, épisode III : La revanche des prototypes

Nous voila arrivés à la dernière mission vers notre quête : devenir un Jedi JavaScript confirmé ne craignant pas le côté obscur de la force. Pour rappel, durant les deux premiers épisodes de cette série d'articles consacrés à l'apprentissage du langage JavaScript dans le but de faire de vous des Jedi, nous avons abordé le côté fonctionnel du langage [1] ainsi que la notion de fermeture [2] (ou "closure" en anglais) qui permet de rendre ces fonctions encore plus polyvalentes et très utiles.

Wassim Chegham
@manekinekko
JavaScript ninja et consultant en technologies Web.

Dans ce dernier épisode nous allons nous attaquer à un dernier pilier du langage JavaScript que sont les prototypes. Probablement que certains d'entre vous sont déjà familiers avec cette notion de prototypes et pensent qu'elle est étroitement liée à la programmation orientée objet. Mais à vrai dire, c'est encore une des nombreuses qualités des fonctions. Oui, j'ai bien dit "fonctions" ! JavaScript a emprunté les prototypes au langage Self [3], un langage orienté objet et dialecte de SmallTalk. Les prototypes sont utilisés pour définir des attributs et des fonctions qui seront appliqués automatiquement à un objet au moment de sa création. Une fois défini, le prototype servira de modèle, pour les objets créés. Vous l'aurez sûrement compris donc, les prototypes jouent le même rôle que les classes dans les langages à base de classes. C'est pour cela qu'en JavaScript, les prototypes sont souvent utilisés par les développeurs pour faire de la programmation orientée objet en mimant la syntaxe des "classes".

Les objets

En JavaScript, toutes les fonctions possèdent une propriété prototype dont la valeur est un objet vide. Cette propriété n'est exploitée que lorsqu'une fonction est invoquée en tant que constructeur, avec le mot-clé new, ce que nous avons déjà détaillé lors du premier épisode. Invoquer une fonction en tant que constructeur revient donc à créer une nouvelle instance. Tentons d'expliquer ce mécanisme d'instanciation pour mieux comprendre le rôle des prototypes.

Instanciation d'objet

En JavaScript, la façon la plus simple pour instancier un objet est comme ceci :

```
var jedi = {};
```

Cette instruction crée un objet vide prêt à l'emploi, et nous pouvons lui définir des propriétés comme ceci :

```
'use strict';

var jedi = {};
jedi.name = 'Luke';
jedi.level = 'padawan';
```

Je pense que je ne vous apprends rien jusques là. Par contre, ceux qui viennent d'un langage orienté objet (au sens strict du terme) préfèreront sûrement avoir un peu plus d'encapsulation et plus de structuration ; c'est-à-dire avoir une sorte de constructeur, une fonction donc, dont le rôle est d'initialiser un objet dans un état connu. Ensuite, utiliser des méthodes pour modifier cet état, au lieu de modifier les attributs directement, risque d'introduire des erreurs et rendre la maintenabilité du code un peu plus complexe. Il est donc préférable d'avoir un mécanisme pour consolider les attributs et méthodes des différents objets instanciés, à un seul endroit. JavaScript propose en effet ce genre de mécanisme, mais il est un peu différent des autres langages. A l'instar de Java, JavaScript utilise le mot-clé new pour instancier de nouveaux objets à travers l'invocation de leurs constructeurs, mais il n'y a pas de définition de classes à proprement parler en JavaScript. Le mot-clé new, une fois appliqué à une fonction, déclenche la création d'un nouvel objet, et, à ce moment-là, les prototypes entrent en jeu. Prenons l'exemple suivant :

```
'use strict';

function Jedi() {}
Jedi.prototype.useForce = function () {
  console.log('I am using the force!');
};

var annikin = Jedi();
var luke = new Jedi();

console.log(annikin instanceof Jedi); //=> false
console.log(typeof annikin.useForce); //=> TypeError: annikin is undefined

console.log(luke instanceof Jedi); //=> true
console.log(typeof luke.useForce); //=> 'function'
```

Analysons ce simple exemple : nous avons défini une fonction Jedi, qui ne fait rien et que nous avons invoquée de deux manières : en tant que fonction et en tant que constructeur. Après la création de la fonction, nous avons ajouté une méthode useForce au prototype de cette fonction.
Dans un premier temps, la fonction a été invoquée normalement et son résultat a été mémorisé dans la variable annikin. Étant donné que la fonction Jedi ne retourne rien, la variable annikin a bien la valeur undefined ; et sans surprise, annikin ne possède pas de méthode useForce. Par contre, en invoquant la fonction Jedi en tant que constructeur avec l'opérateur new, le résultat est tout autre. Il se trouve que cette fois-ci un nouvel objet a été créé et positionné comme étant le contexte d'exécution de la fonction, et le résultat retourné par le constructeur est la référence vers cet objet. L'inspection de l'instance luke nous prouve bien que cette instance de Jedi et l'objet luke possèdent bien la méthode useForce, récupérée depuis son prototype Fig.1.
Ce premier exemple démontre que le rôle des prototypes est bien de servir de modèle aux objets instanciés. Juste le fait d'attacher la méthode useForce au prototype de Jedi l'a rendu disponible dans l'instance créée.
Voici un schéma illustrant les liens entre une instance, son constructeur et le prototype : Fig.2.
Nous avons vu que l'opérateur new a créé un nouvel objet qui sert par la suite de contexte d'exécution de la fonction (le constructeur), et nous avons également vu qu'il était possible d'attacher des propriétés au prototype de cette fonction. Puisque la fonction possède un contexte, nous

pouvons également attacher des propriétés au constructeur directement via le paramètre this. Étudions ce cas de près :

```
'use strict';

function Jedi() {

 this.useForce = function () {
  return 'I am the Instance';
 };
}

// 1) nous attachons la méthode "useForce" au prototype
Jedi.prototype.useForce = function () {
 return 'I am the Prototype';
};

// 2) nous créons une instance
var luke = new Jedi();
console.log(luke.useForce()); //=> 'I am the Instance'
```

Comme pour l'exemple précédent, en (1) nous avons attaché une méthode useForce au prototype du constructeur. De plus, nous avons ajouté une méthode portant le même nom au sein du constructeur. Les deux méthodes retournent un résultat différent pour que nous puissions savoir laquelle a été appelée.

En (2), après avoir créé une instance luke, et invoqué la méthode useForce, nous constatons que c'est bien la méthode définie dans le constructeur qui a été invoquée. L'ordre d'initialisation des propriétés est donc très important et suit la logique suivante :

- Les propriétés sont **attachées** à l'instance de l'objet depuis le prototype ;
- Les propriétés sont **ajoutées** à l'instance de l'objet depuis le constructeur.

Autrement dit, les propriétés ajoutées dans le constructeur passent toujours avant celles attachées au prototype. La raison est que le this, autrement dit le contexte, dans le constructeur représente l'instance elle-même Fig.3. Étudions un autre cas afin de mieux comprendre le lien entre les prototypes et les instances d'objets. Prenons le code de l'exemple précédent et modifions-le un peu pour avoir ceci :

```
'use strict';

function Jedi() {

 this.useForce = function () {
  return 'I am the Instance';
 };
}

// 1) nous créons une instance...
var luke = new Jedi();

// 2) ... ensuite nous attachons la méthode "useForce"
Jedi.prototype.useForce = function () {
 return 'I am the Prototype';
};

console.log(luke.useForce()); //=> 'I am the Instance'
```

Dans ce code, nous avons échangé l'ordre de création de l'instance (1) et celui d'attachement de la méthode useForce au prototype (2). Pourtant, si nous invoquons cette méthode, elle est bien présente, et son résultat est celui attendu. Comment se fait-il ?

Nous aurions pu penser que les propriétés attachées au prototype sont simplement copiées vers l'objet au moment de sa création, et qu'ensuite tout changement effectué sur le prototype après la construction de l'objet ne serait pas reporté sur l'instance créée. En réalité les propriétés du prototype ne sont pas du tout copiées, c'est plutôt le prototype lui-même qui est attaché à l'objet construit. En JavaScript, chaque objet possède une propriété appelée constructor qui référence le constructeur qui a été invoqué pour créer l'objet en question. Comme le prototype est une propriété du constructeur, chaque objet sait donc comment accéder à son prototype. Vérifions cela avec cet exemple :

```
console.log(luke.constructor.toString());
/*
"function Jedi(){

 this.useForce = function(){
  return 'I am the Instance';
 };

}"
```

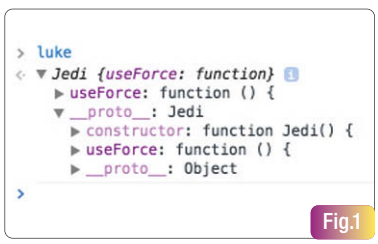


Fig.1

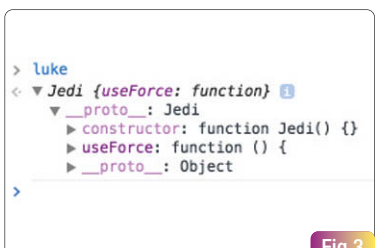


Fig.3

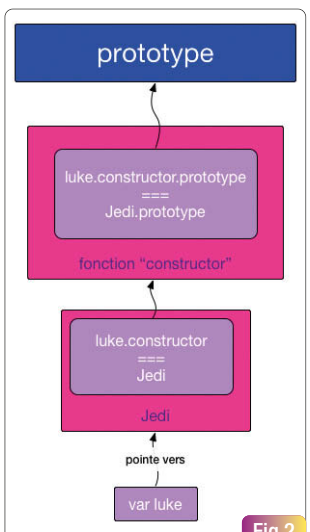


Fig.2

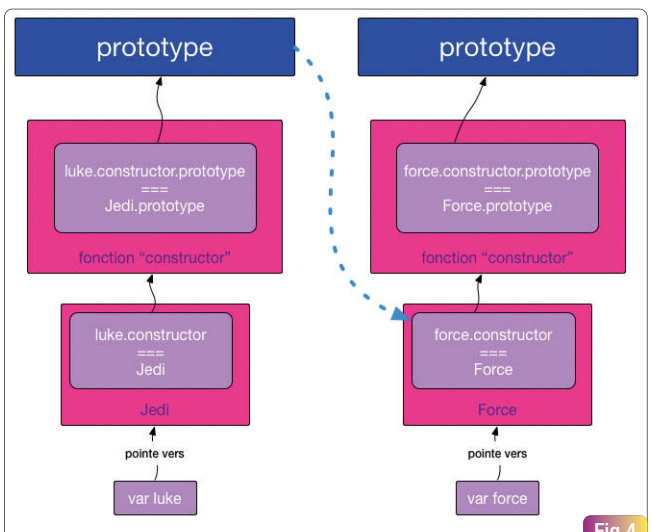


Fig.4

```
*/
console.log(luke.constructor.prototype.useForce.toString());
/*
"function (){
 return 'I am the Prototype';
}"
*/
```

Nous avons bien accès au constructeur de l'instance luke ainsi qu'à son prototype et donc à toutes les propriétés attachées au prototype ; ce qui permet d'expliquer pourquoi tout changement effectué sur le prototype après la création de l'objet est automatiquement présent sur ce dernier.
Je vous laisse imaginer l'étendue de ce que nous offre ce genre de fonctionnalités. Nous pouvons faire des librairies que les utilisateurs pourront enrichir, même après que tous les objets auront été instanciés.
Maintenant que la notion de prototype n'a plus de secret pour vous, nous allons introduire une autre notion intimement liée aux prototypes : la chaîne des prototypes.

La chaîne des prototypes et l'héritage

Afin d'expliquer ce qu'est la chaîne des prototypes, prenons l'exemple suivant :

```
'use strict';

function Force() {}
Force.prototype.useForce = function () {
 return 'I am the Force to be used.';
};

function Jedi() {}
Jedi.prototype = {
 useForce: Force.prototype.useForce
};

var luke = new Jedi();
console.log(luke.useForce()); //=> 'I am the Force to be used.'
console.log(luke instanceof Jedi); //=> true
console.log(luke instanceof Force); //=> false
```

Le prototype d'une fonction étant un simple objet, il existe plusieurs façons de lui attacher des propriétés. Dans l'exemple précédent, nous avons défini une Force et un Jedi, et puisqu'un Jedi est le seul individu apte à maîtriser la Force, et l'utilise pour faire du bien, nous allons faire en sorte que le Jedi hérite des attributs de Force. Nous faisons cela en copiant la méthode useForce depuis le prototype de Force vers une méthode useForce dans le prototype de Jedi.
En testant la présence de la méthode useForce, ainsi que la nature du type de luke, nous réalisons que luke a la faculté maintenant d'utiliser la Force, mais il ne l'incarne pas, il n'est pas la Force ! Si nous souhaitons qu'il devienne la Force, nous devrions copier toutes les propriétés de Force vers le prototype de Jedi, une par une ! Sinon, plus simplement, il suffit que Jedi hérite des propriétés de Force. Jusque-là je ne vous apprends rien. Cependant, en JavaScript, nous allons nous baser sur ce que l'on appelle la chaîne des prototypes pour bénéficier de l'héritage entre les objets.
La "bonne" façon de réaliser cette chaîne est de créer une instance d'un objet et de l'utiliser en tant que prototype d'un autre objet :

```
'use strict';

function Force() {}
Force.prototype.useForce = function () {
 return 'I am the Force to be used.';
};

function Jedi() {}
Jedi.prototype = new Force();

var luke = new Jedi();
console.log(luke.useForce()); //=> 'I am the Force to be used.'
console.log(luke instanceof Jedi); //=> true
console.log(luke instanceof Force); //=> true
```

Maintenant, luke incarne la Force ! Voici ce que cela donne sous forme de schéma : Fig.4. A titre d'exemple, voici la "mauvaise" façon de réaliser une chaîne des prototypes...

```
Jedi.prototype = Force.prototype;
```

Vous allez me dire, après tout, les deux prototypes sont des objets, pourquoi ne pas faire une simple affectation ? La raison est simple : les objets sont copiés par référence en JavaScript; au moment de l'affectation, les deux prototypes référenceront le même objet en mémoire. Modifier le prototype de Jedi revient à modifier celui de Force, ce qui peut conduire à des effets indésirables :

```
'use strict';

function Force() {}
Jedi.prototype.toString = function () {
 return 'I am the Force';
};

function Jedi() {}
Jedi.prototype = Force.prototype;
Jedi.prototype.toString = function () {
 return 'I am a Jedi';
};

var luke = new Jedi();
var force = new Force();
console.log(luke instanceof Jedi); //=> true
console.log(luke instanceof Force); //=> true
console.log(luke + ''); //=> 'I am a Jedi'
console.log(force + ''); //=> 'I am a Jedi' <=== OH OH !!!!
```

Vous l'aurez compris donc, cette pratique est à proscrire !
Après avoir étudié les prototypes et exploré la flexibilité et la puissance offertes par les prototypes, ainsi que la chaîne des prototypes, passons maintenant à la suite de l'épisode et tentons de mettre en pratique cette fonctionnalité pour implémenter des "classes" (comme dans les autres langages du type C++ et Java).

Implémentation d'une "classe"

Je constate souvent que pas mal de développeurs, particulièrement ceux qui viennent des langages objet à base de classes (Java ou C++ typiquement), préfèrent avoir une sorte d'abstraction leur permettant de simplifier l'implémentation de ce qu'ils appellent des "classes".
En JavaScript, s'il y a deux choses à retenir en POO, ce sont :

- La notion de "classe" n'existe pas,

- Il n'y a pas d'héritage de classes, mais un héritage de prototypes — puisque les classes n'existent pas !

Mais grâce aux prototypes qui nous permettent d'enrichir le langage en le rendant beaucoup plus flexible, il est possible d'avoir :

- Une sorte de syntaxe pour implémenter des constructeurs et des prototypes,
- Un mécanisme simple pour réaliser l'héritage des prototypes,
- Un moyen d'accéder à des méthodes surchargées par le prototype.

Je tiens à préciser tout de même que ce que nous allons voir dans la suite de l'article est simplement un sucre syntaxique offert par le langage pour permettre aux développeurs qui tiennent tant aux "classes" de pouvoir "imiter" ce fonctionnement en JavaScript.

Je vous recommande donc lorsque vous développez en JavaScript de concevoir vos applications en raisonnant en "prototypes" et non pas en "classes", afin d'exploiter au maximum la puissance du langage. D'ailleurs, ceci est vrai pour n'importe quel autre langage.

Regardons maintenant comment peut-on faire des "classes" en JavaScript...Mais avant de passer à la suite, sachez qu'il existe plusieurs façons d'écrire des "classes" en JavaScript, ce qui est normal puisque comme je l'ai précisé juste avant, JavaScript ne gère pas les "classes" et la conséquence directe de cela est que l'on va retrouver différentes implémentations dans la littérature. Pour ma part je vais vous présenter une des implémentations, la plus simple, à mon avis.

"Classe" et héritage en ECAMScript 5 (version actuelle de JavaScript)

Reprenons l'exemple de code de notre Jedi :

```
'use strict';

function Jedi(name) {
 this.name = name;
}

Jedi.prototype.toString = function () {
 return 'I am ' + this.name;
};

var luke = new Jedi('luke');
console.log(luke.toString()); //=> 'I am luke'
```

Essayons maintenant de coder une abstraction qui va nous permettre de créer des objets spécialisés et bénéficier d'un héritage au passage...

```
'use strict';

function Klass(parent, child) {
 if (!child) {
  return parent;
 }

 //1)
 child.prototype = new parent();

 //2)
 child.prototype.constructor = child;

 //3)
 return child;
}
```

A noter que ceci reste une implémentation naïve, à titre d'exemple, vous ne devriez pas l'utiliser dans vos applications en production.

Expliquons ce que fait ce petit bout de code :

- Nous réalisons un héritage de prototypes à ce niveau, comme vu précédemment,
- Nous corrigeons le constructeur de "child" pour qu'il pointe sur celui de "child", car à cause de l'étape précédente, il pointe vers celui de "parent",
- Enfin, nous retournons le constructeur "child" pour qu'il puisse être instancié.

Voilà ! Vous avez maintenant une API Klass vous permettant de coder des "classes". Voici comment nous l'utilisons :

```
'use strict';

var force = Klass(function Force(name) {
 this.me = name || 'Force';
 this.print = function () {
  return this.me + ' Force';
 };
});
var jedi = Klass(force, function Jedi(name) {
 this.me = name || 'Jedi';
 this.print = function () {
  return this.me + ' POWA!!';
 };
});

var yoda = new jedi('Yoda');
console.log(yoda instanceof jedi); //=> true
console.log(yoda instanceof force); //=> true
console.log(yoda.print()); //=> Yoda POWA!!
```

Si vous préférez utiliser ce genre de sucre syntaxique, sachez qu'il existe une multitude de micro librairies JavaScript dédiées à cet usage, que vous allez pouvoir utiliser en production [4]. Cependant, j'ai peur que ces librairies ne soient obsolètes rapidement. La raison ? La sortie prochaine de la future version de JavaScript, ECMAScript 6, qui apporte énormément de changement et de fonctionnalités qui manquaient tant au langage. Parmi elles, on retrouve une nouvelle syntaxe pour écrire des "classes" ; mais ne vous méprenez pas, cela reste uniquement un sucre syntaxique proposé par cette nouvelle version. En interne, les prototypes règnent toujours en maître.

"Classe" et héritage en ECAMScript 6 (prochaine version de JavaScript)

Voici à quoi va ressembler la nouvelle syntaxe permettant d'écrire de la POO en JavaScript tout en faisant plaisir aux développeurs adeptes des "classes" :

```
'use strict';

class Force {
 constructor(name = 'Force'){
  this.me = name;
 }
 print(){
  return this.me + ' Force';
 }
}
```

```
class Jedi extends Force {
 constructor(name = 'Jedi'){
  super();
  this.me = name;
 }
 print(){
  return this.me + ' POWA!!'
 }
}

var yoda = new Jedi();
console.log(yoda instanceof Jedi); //=> true
console.log(yoda instanceof Force); //=> true
console.log(yoda.print()); //=> Yoda POWA!!
```

Tout simplement ! Vous pouvez voir la version transpilée en ECMAScript 5 sur cette page Web [5] en utilisant le célèbre transpileur [6] Babel (anciennement 6to5).

Conclusion

Grâce à cette série d'articles consacrés à l'apprentissage des fondamentaux du langage JavaScript, nous avons appris et surtout compris les trois piliers du langage : les objets à travers la POO, les fonctions et les closures. Nous avons donc vu quelle était l'importance pour un développeur débutant en JavaScript d'apprendre vraiment ce langage afin d'en tirer le meilleur et devenir un Jedi.

Prenez le temps d'apprendre, pratiquer et surtout comprendre ces trois piliers, et vous verrez que JavaScript n'aura plus de secrets pour vous. Ceci est d'autant plus important et essentiel puisque dans les mois à venir, le langage va connaître une évolution majeure depuis sa naissance en 1995. C'est le moment idéal donc de bien maîtriser les bases du langage car cette évolution, annoncée pour Juin 2015, va apporter tout un ensemble d'améliorations et de fonctionnalités toutes aussi puissantes et utiles les unes les autres, et que tout Jedi JavaScript va devoir adopter.

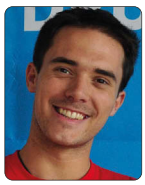
Que la force d'ECMAScript soit avec vous, fidèles Jedi.

[1] Article paru dans le numéro 183 de Programmez
[2] Article paru dans le numéro 184 de Programmez
[3] http://en.wikipedia.org/wiki/Self_(programming_language)
[4] http://microjs.com/#class
[5] http://bit.ly/JavaScriptPourLesJedisEp3-es6-class
[6] http://bit.ly/DefineTranspiler

Votre boutique en ligne avec Drupal Commerce & Platform.sh

1ère partie

Dans cet article, vous allez lancer, en quelques minutes, votre boutique en ligne. C'est possible grâce à des outils appropriés et en utilisant une approche "lean", qui préconise un lancement du produit très tôt avec des itérations continues.



Augustin Delaporte
augustin@commerceguys.com
@GussTissier
Product Delivery Manager chez Commerce Guys
En charge du développement de Commerce Kickstart et Platform.sh.

Drupal Commerce est un framework de développement d'applications e-commerce totalement intégré à Drupal, et utilisé par plusieurs dizaines de milliers de marchands à travers le monde. Platform.sh est la solution de développement et d'hébergement en intégration continue. Elle vous permet de réduire le délai de lancement de vos applications d'une façon considérable, et de tester vos nouvelles fonctionnalités en conditions réelles, afin de pouvoir itérer sur votre site de production aussi fréquemment que souhaité.

Note : *Pour suivre ce tutoriel, il est nécessaire de savoir utiliser Git. Une première expérience avec Drupal est également très utile, ainsi que la connaissance de Drush, son outil en ligne de commande.*

C'EST PARTI

Votre produit

Dans cet article, vous proposez à vos clients de payer pour qu'un étranger les appelle un jour aléatoire de la semaine prochaine pour les motiver et leur dire combien ils sont géniaux.

Démarrer

Commencez par créer un projet sur platform.sh. Suivez les étapes de configuration du projet et choisissez de démarrer avec *Drupal 7* Fig.1.
A la fin de l'initialisation, votre site Drupal est déployé et accessible depuis votre navigateur.
Depuis l'interface de Platform.sh, copiez la commande CLONE de Git et collez cette commande dans votre terminal, en ajoutant "motivation-call" pour avoir un joli nom de dossier. Notez que votre commande SSH sera différente car elle dépend de l'identifiant de votre projet :

```
$> git clone --branch master zc36havdfoqks@git.eu.platform.sh:zc36havdfoqks.git motivation-call
$> cd motivation-call
$> ls -al
.git/
.platform/
.platform.app.yaml
libraries/
modules/
project.make
themes/
```

Note : *Platform.sh a créé un nouveau répertoire Git pour votre projet, dont la branche master a été initialisée à partir du répertoire public suivant :* https://github.com/platformsh/platformsh-examples/tree/drupal/7.x.

Créer un profil d'installation

Vous allez construire un profil d'installation pour développer votre boutique en ligne. C'est une méthode très efficace qui nécessite néanmoins un peu d'effort au démarrage. L'objectif est d'exporter la totalité de la configuration de votre site, habituellement stockée en base de données, dans des fichiers. Ainsi, vous pouvez réinstaller votre site à chaque instant du développement du projet et traquer tous les changements de la configuration via Git.

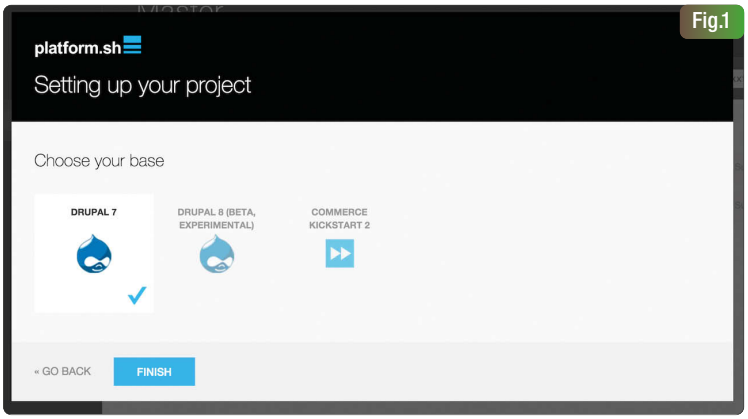


Fig.1

Note : *Dans cet article, nous n'utilisons pas la distribution Commerce Kickstart (https://www.drupal.org/project/commerce_kickstart) qui doit être considérée comme un outil d'apprentissage et d'exemple plutôt qu'une base de démarrage pour un nouveau projet.*

Commencez par ajouter les dépendances de votre boutique en éditant le fichier *project.make* (*vous pouvez supprimer tout le contenu initial*) :

```
api = 2
core = 7.x

defaults[projects][subdir] = contrib

; Contributed modules
projects[admin_menu][version] = 3.0-rc5
projects[addressfield][version] = 1.0
projects[ctools][version] = 1.7
projects[commerce][version] = 1.11
projects[commerce_features][version] = 1.0
projects[commerce_migrate][version] = 1.1
projects[entity][version] = 1.6
projects[entityreference][version] = 1.0
projects[features][version] = 2.4
projects[jquery_update][version] = 2.5
projects[migrate][version] = 2.7
projects[migrate_extras][version] = 2.5
projects[rules][version] = 2.9
projects[views][version] = 3.10

; Contributed themes
projects[bootstrap_business][version] = 1.1
```

Ajoutez un nouveau fichier *project-core.make* pour Drupal core, qui sera exécuté avant l'exécution de votre *project.make*.

```
core = 7.x
api = 2

; Drupal core version.
projects[drupal][version] = 7.35
projects[drupal][patch][] = "https://drupal.org/files/issues/install-redirect-on-empty-database-728702-36.patch"
```

Note : *Au moment de pousser vos modifications via Git, Platform.sh va détecter la présence de vos fichiers .make et automatiquement lancer la commande drush make dont la sortie sera visible directement dans votre terminal.*

Un profil d'installation est constitué au minimum d'un *.profile* et d'un *.info* :

- Le *.info* définit les modules, thèmes et librairies à activer au moment de l'installation du profil.
- Le *.profile* contient, dans notre cas, simplement un *hook_install* qui sera exécuté pendant l'installation du profil. Nous avons simplifié ce hook au maximum, mais il est possible de configurer un grand nombre de paramètres dès l'installation (*voir le hook_install du profil Standard de Drupal ou de Commerce Kickstart*).

Créez le fichier *motivation_call.info*:

```
name = Motivation Call
description = A store to purchase motivation calls.
core = 7.x

; Core modules
dependencies[] = image

; Contributed modules
dependencies[] = admin_menu_toolbar
dependencies[] = views_ui
dependencies[] = jquery_update

; Commerce modules
dependencies[] = commerce
dependencies[] = commerce_ui
dependencies[] = commerce_cart
dependencies[] = commerce_checkout
dependencies[] = commerce_customer
dependencies[] = commerce_line_item
dependencies[] = commerce_order
dependencies[] = commerce_order_ui
dependencies[] = commerce_payment
dependencies[] = commerce_payment_ui
dependencies[] = commerce_payment_example
dependencies[] = commerce_price
dependencies[] = commerce_product
dependencies[] = commerce_product_ui
dependencies[] = commerce_product_pricing
dependencies[] = commerce_product_pricing_ui
dependencies[] = commerce_product_reference
dependencies[] = commerce_tax
dependencies[] = commerce_tax_ui
```

Note : *Vous pouvez noter à quel point Drupal Commerce est modulaire. Chaque module vient avec une interface graphique (UI) séparée du fonctionnel. De plus, chaque fonctionnalité peut être activée séparément, ce qui implique que vous choisissiez seulement ce dont vous avez besoin, et que vous puissiez implémenter vos propres fonctionnalités si vous ne souhaitez pas utiliser l'implémentation par défaut (panier, tunnel d'achat...). Pour notre lean startup, nous allons conserver toutes les fonctionnalités par défaut.*

Créez également le fichier *motivation_call.profile* :

```
<?php

/**
 * Implements hook_install()
 */
function motivation_call_install() {
  // Enable Bootstrap Business theme.
  theme_enable(array('bootstrap_business'));
  variable_set('theme_default', 'bootstrap_business');

  // Set the admin theme.
  db_update('system')
    ->fields(array('status' => 1))
    ->condition('type', 'theme')
    ->condition('name', 'seven')
    ->execute();
  variable_set('admin_theme', 'seven');
}
```

Vous pouvez maintenant pousser vos modifications via Git. :

```
$> git add –all
$> git commit -m "Initial commit of the profile."
$> git push
```

Vous verrez dans votre terminal que Platform.sh est en train de télécharger les dépendances de votre profil Fig.2.
En accédant à votre environnement *Master* dans votre navigateur, vous verrez votre profil disponible Fig.3.

Automatiser l'installation du profil

Platform.sh propose des *hooks de déploiement* qui vous permettent d'interagir sur le déploiement de votre site au moment de pousser vos modifications via Git.
Pendant le développement de votre site, il est intéressant de relancer l'installation du site à chaque *git push* afin de s'assurer que toute la configuration est bien exportée.
Pour cela, éditez simplement le fichier *.platform.app.yaml* :

```
augustin /tmp/motivation-call [master] git push
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.29 KiB | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)

Validating submodules.
Validating configuration files.
Processing activity: **Augustin Delaporte** pushed to **Master**
    Found 1 new commit.

    Building application 'php' with toolstack 'php:drupal' (tree: 85e361c)
      Installing build dependencies...
        Installing php build dependencies: drush/drush

      Making profile `motivation_call`...
      Found `project.make`, building it...
        Executing `drush -y make --cache-duration-releasexml=300 --concurrency=8 --no-core --contrib-destination=. project.make .`...
          Beginning to build project.make.                                    [ok]
            >> features-7.x-2.4 downloaded.                                   [ok]
            >> commerce_features-7.x-1.0 downloaded.                          [ok]
            >> Project entityreference contains 2 modules: entityreference_behavior_example, entityreference.
            >> entityreference-7.x-1.0 downloaded.                            [ok]
            >> Project entity contains 2 modules: entity, entity_token.
            >> entity-7.x-1.6 downloaded.                                     [ok]
```

Fig.2

```
hooks:
  deploy: |
    cd public
    drush site-install motivation_call -y --account-pass=admin --site-name="Motivation Call Store"
```

Si vous poussez cette modification via Git, votre profil sera installé automatiquement via Drush à la fin du déploiement.
La sortie des *hooks de déploiement* est stockée dans le fichier */var/log/deploy.log* accessible en SSH sur le serveur :

```
$> ssh -t zc36havdfoqks-master@ssh.eu.platform.sh "cat /var/log/deploy.log"

[2015-03-20 16:04:40.896193] Launching hook 'cd public
drush site-install motivation_call -y --account-pass=admin --site-name="Motivation Call Store"'

You are about to DROP all tables in your 'main' database. Do you want to continue? (y/n): y
Starting Drupal installation. This takes a few seconds ...                [ok]
Installation complete.  User name: admin  User password: admin            [ok]
```

Configurer la boutique

Maintenant que votre profil est prêt à l'emploi, vous allez ajouter un nouveau type de produit et l'exporter dans une *feature* (ou fonctionnalité).
Une *feature* est une collection d'entités Drupal qui satisfont un cas d'utilisation (*boutique, blog, portfolio...*). Ici, nous utilisons le module Features (https://www.drupal.org/project/features) pour exporter la configuration de notre boutique dans une *feature*.
Connectez-vous sur votre site (/user) avec le login: *admin/admin*. Créez un type de produit *Motivation Call* (/admin/commerce/products/types/add) et ajoutez le champ Image existant. Conservez toutes les valeurs par défaut **Fig.4**.

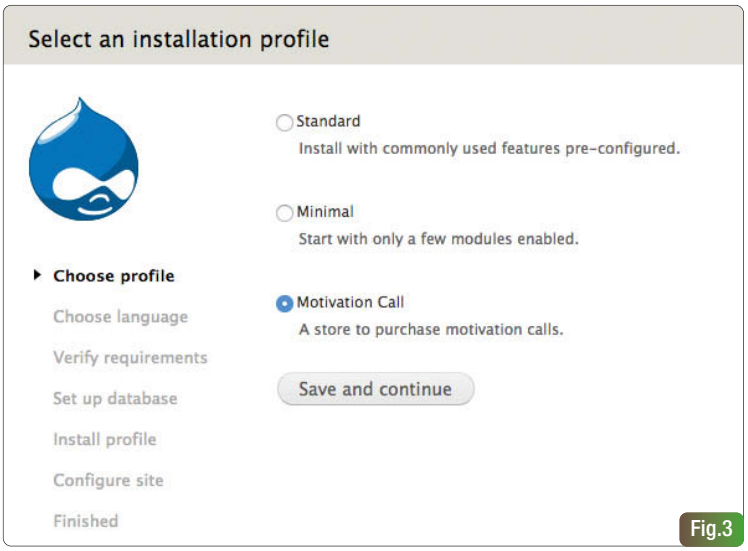


Fig.3

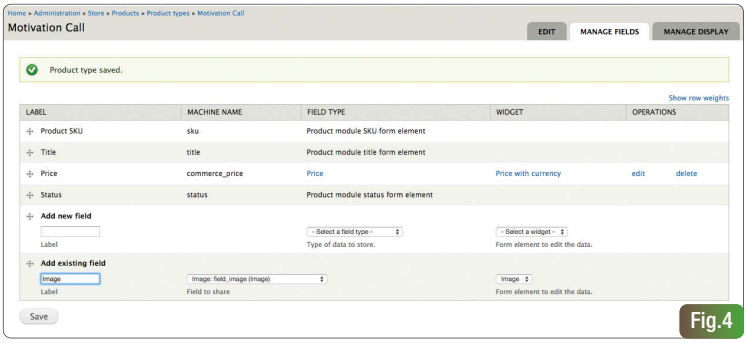


Fig.4

Ajoutez également un taux de TVA de 20% (/admin/commerce/config/taxes/rates/add) que nous allons également exporter dans notre *feature* :

- Titre: *TVA 20*
- Type *VAT*
- Valeur: *0.2* **Fig.5**.

Créez une nouvelle *feature* appelée: *Store Configuration* (admin/structure/features/create), et ajoutez les composants que vous venez de créer : *Commerce Product Types* et *Commerce Tax Rates*. Vous pouvez maintenant télécharger la *feature* et l'extraire dans le répertoire *modules* de votre projet **Fig.6**.
Ajoutez la nouvelle dépendance dans votre fichier *motivation_call.info* pour activer votre *feature* pendant l'installation du profil.

```
; Features dependencies
dependencies[] = store_configuration
```

Poussez vos modifications avec Git et vérifiez que votre *feature* a bien été activée.

La suite dans le numéro 186.

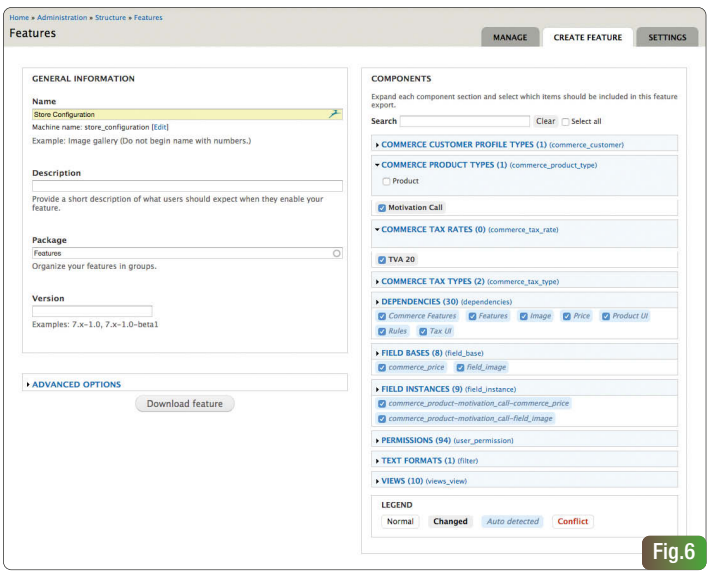


Fig.6

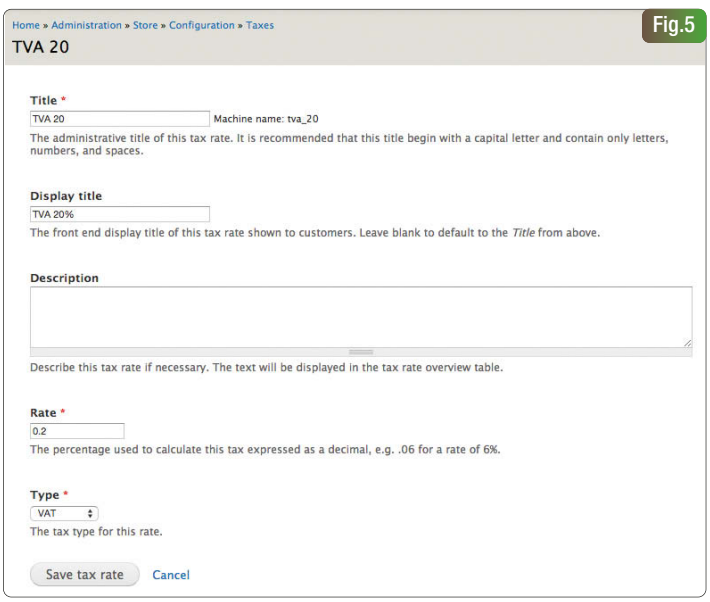
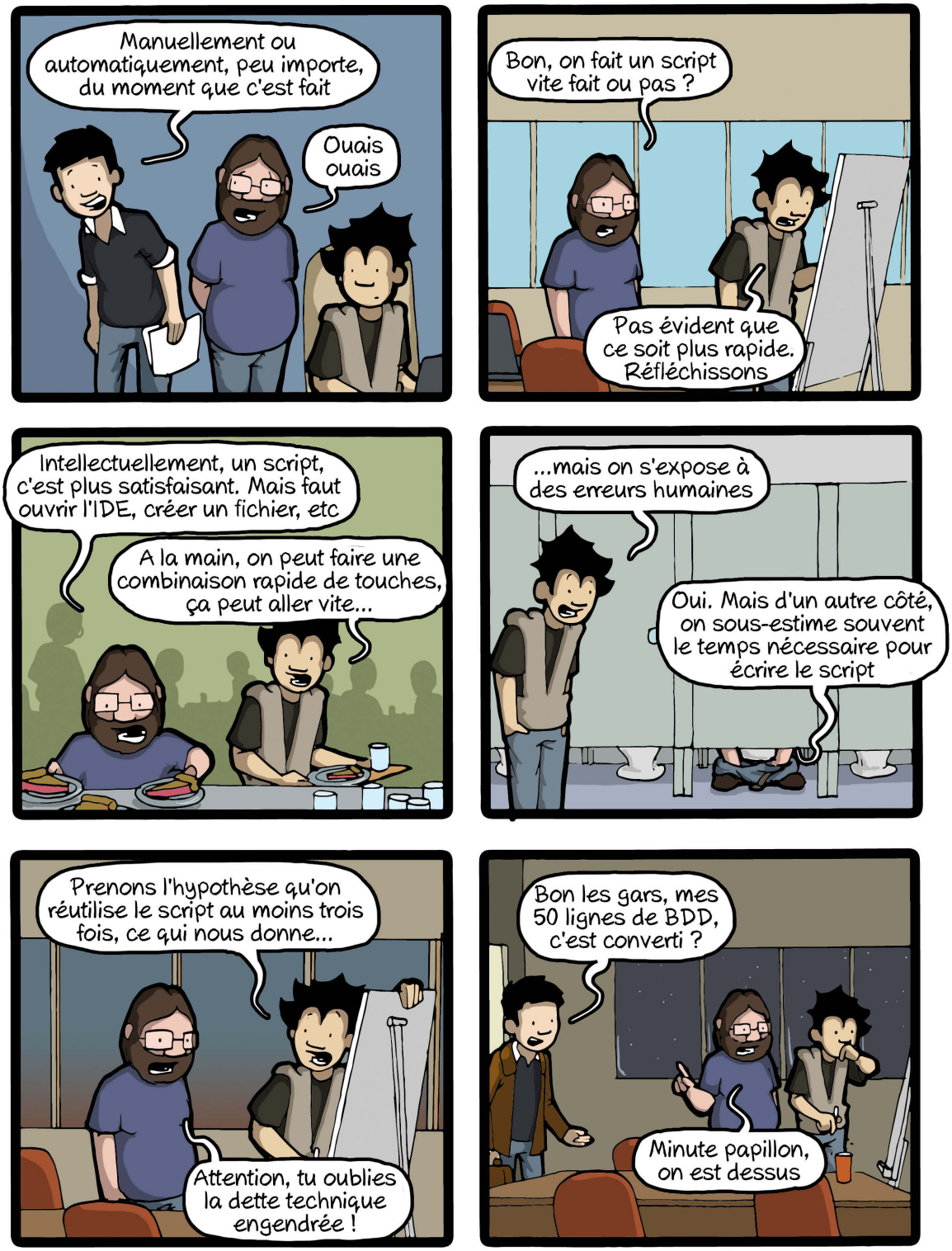


Fig.5



Confier une tâche répétitive à un codeur

www.commitstrip.com

Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € Autres pays : nous consulter.
PDF : 30 € (Monde Entier) souscription sur www.programmez.com

Une publication **Nefer-IT**
7 avenue Roger Chambonnet
91220 Brétigny sur Orge
redaction@programmez.com
Tél. : 01 60 85 39 96

Directeur de la publication & rédacteur en chef : François Tonic

Ont collaboré à ce numéro : S. Saurel.

Secrétaire de rédaction : Olivier Pavie
Experts : E. Margraff, A. Moreau, K. Albrecht, S. Ollivier, C. Autexier, T. Templier, C. Pichaud, C. Villeneuve, G. Renard, E. Vernié, F. Zaninotto, A. Moreau, F. Fadel, M. Lorber, J. De Oliveira, F. Bellahcene, Q. Ochem, F. Bensusan, S. Cordonnier, S. Sibué, Patrick-André Marendat, S. Ben Fredj, T. Guerin, S. Cordonnier, M. Fery, M. Frappat, W. Chegham, A. Delaporte

Crédits couverture : © Maxiphoto

Maquette : Pierre Sandré

Publicité : PC Presse,
Tél.: 01 74 70 16 30, Fax : 01 41 38 29 75
pub@programmez.com

Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.

Marketing et promotion des ventes :
Agence BOCONSEIL - Analyse Media Etude

Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Contacts
Rédacteur en chef : ftonic@programmez.com
Rédaction : redaction@programmez.com
Webmaster : webmaster@programmez.com
Publicité : pub@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1215 K 78366 - ISSN : 1627-0908

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

Ou encore sur votre tablette

ikoula
HÉBERGEUR CLOUD

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

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

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

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

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

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