SPÉCIAL 20 ANS | SPÉCIAL 20 ANS | SPÉCIAL 20 ANS | SP

programmez.com

[Programmez!]

Le magazine des développeurs

218 MAI 2018

NUMÉRO SPÉCIAL 20 ANS

Q# : la programmation quantique à votre portée

Le top 10 des erreurs JavaScript

Réalité augmentée

La machine de Turing

20 ans de technologies : les tops, les flops

© Lenovo

Java 10
Une version mineure

Infrastructure as Code
Pourquoi et comment coder son infrastructure ?

M 04319 - 218S - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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

NUTANIX .NEXT ON TOUR

6 Countries. 16 Cities.

ONE OS. ONE CLICK.
ANY CLOUD.

Après avoir parcouru l'Europe du Sud et le Moyen-Orient avec une présence dans 6 pays et 16 villes, Nutanix achève son .Next on Tour à Paris le 7 juin 2018. Ce jour sera l'occasion de partager notre vision et notre stratégie. Vous en profiterez pour rencontrer nos équipes et nos partenaires technologiques.

Seront présents avec nous sur cette journée, Mr Sudheesh Nair - Président de Nutanix et Mr Raja Mukhopadhyay - VP Product Management.

Paris

Inscrivez-vous via le lien ci-dessous:

www.nutanixtour.com/paris

ou en scannant ce QR Code :

PLATINUM

VEEAM
VERITAS

GOLD

f5
HYCU
paloalto NETWORKS
Mellanox TECHNOLOGIES
SUSE

SILVER

Bitdefender
Global Knowledge
CITRIX
rubrik

EDITO

1998, là, faut oser !

DOS, disquettes, modems, écrans à tube, téléphones même pas tactiles, portables qui pèsent 3-4 kg, VB… Cela fait peur non ? Et pourtant c'était notre quotidien. Les interfaces Gnome et KDE étaient à peine naissantes. Windows XP n'existait pas encore, ni MacOS X ! On parlait de Wintel partout. Apple était au bord de la faillite. Microsoft régnait en maître et Linux commençait juste à faire sa place sur les serveurs. Et Internet n'était pas aussi répandu qu'aujourd'hui, et encore moins le haut débit ou la 3G (qui n'était qu'un rêve).

En 2018, on parle IoT, Cloud computing, le tout connecté, d'intelligence artificielle, de quantique, de réalité virtuelle en mode web. Le téléphone est aussi puissant qu'un ordinateur. Le moindre logiciel pèse 500 Mo, parfois dépasse 20 Go (installation complète de Visual Studio par exemple). Fini les logiciels qu'on achète en boîte, on passe par un App Store. On parle d'écran 4K, 5K et bientôt 8K.

En 20 ans, beaucoup de choses ont changé pour l'utilisateur mais aussi pour le développeur. Et c'est ça qui est vraiment intéressant pour nous. A Programmez! nous avons la chance de voir mois après mois les évolutions, les nouveautés. On s'étonne du succès de certaines technologies et de l'échec d'autres. Depuis 20 ans, on nous dit que Cobol, le mainframe, le C++, c'est fini. Mais ils sont toujours là et pour longtemps !

Depuis les origines de Programmez!, c'est cette passion, cette envie de partager qui nous anime.

20 ans, 218 numéros, +18 000 pages, des centaines de contributeurs, Programmez! est une belle aventure. Nous avons rencontré des gens géniaux. Certain(e)s d'entre vous nous lisent depuis le n°1, d'autres depuis ce mois-ci !

Nous tenons aussi à remercier Jean Kaminsky, éditeur de la revue durant 15 ans, qui a soutenu, vaille que vaille, le magazine.

François Tonic
ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !
Programmez! #219, dès le 1er juin 2018

Xamarin : stop ou encore ?
Où en est la solution multiplaforme de Microsoft ? Qu'en pense la communauté ? Quelles sont les nouveautés les plus récentes ?

La réalité augmentée par la pratique :
exemples concrets d'usages et de développements

Dossier Python

Adobe abandonne son outil de création web sans code, Muse. Après mai 2019, plus aucune mise à jour.

Tesla va-t-il survivre à 2018 ? Certains analystes en doutent ! Il faut +8 milliards $ pour continuer l'activité, rembourser les dettes et surtout améliorer la production des voitures qui plafonne a à peine 1 000 exemplaires / mois !

5 saisons seraient prévues pour la série sur le Seigneur des Anneaux. Amazon joue très gros sur ce projet. L'investissement pourrait être proche d'un milliards $, du jamais vu pour une série. Pour le moment aucun détail sur les histoires qui seront abordées, l'implication (ou non) de Peter Jackson, le nombre d'épisodes par saison.
La production doit démarrer dans les 2 ans…

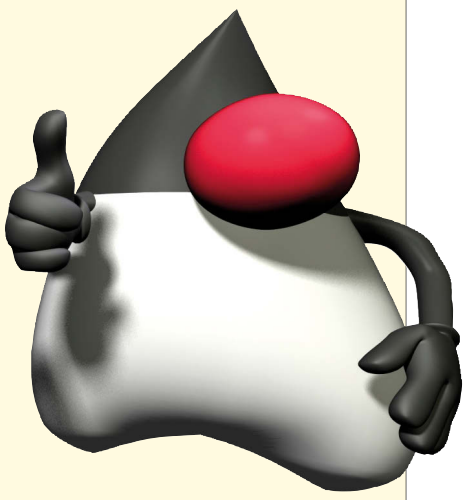
Terminator 6 : début du tournage prévu à partir de juin si tout va bien. Ce sera la suite directe de Terminator 2…

Intel a connu tout début avril une belle baisse en bourse. La raison ? La rumeur, ou plutôt le retour d'une rumeur, sur la possibilité qu'Apple puisse faire la bascule sur les processeurs ARM, à partir de 2019. Ce serait un mouvement stratégique important : la maîtrise de l'évolution des processeurs et une optimisation de ceux-ci. Il faut dire qu'Apple possède une équipe processeur très performante. Le passage à ARM est une vieille rumeur.
Apple possède les compétences pour réaliser la bascule mais elle ne peut se faire que progressivement (selon les gammes de machine) et il faut préparer plusieurs mois à l'avance les développeurs. La date de 2020 est même avancée pour le début de l'opération.

MICROSOFT : CLOUD ET IA, ET WINDOWS ?

Depuis l'arrivée de Satya Nadella, Microsoft a subi plusieurs réorganisations. Celle de ce printemps est sans doute la plus importante. L'éditeur mise tout sur le Cloud Computing (ce n'est pas nouveau) et sur l'IA. Sur la partie IA, Microsoft veut saisir le marché naissant sur les outils et les plateformes.
Depuis l'arrivée du remplaçant de Ballmer, Microsoft a pris un véritable tournant sur les services et est allé où les clients sont. Ils utilisent iOS et Android, pas de problème, on sort des apps dessus, quitte à délaisser leur propre plateforme.
Le Cloud Computing s'impose. Microsoft y va à marche forcée avec le Cloud First. Et aujourd'hui, Azure, et l'ensemble des piles cloud, est la plateforme sans doute la plus complète du marché.
Depuis plusieurs années, nous pensons que Microsoft ne doit plus penser Windows (donc OS centrique) mais logiciels et services. Le système n'est finalement qu'un moyen d'accès. Et le déclin du PC confirme cette tendance au profit de l'accès mobile. L'échec de Windows Phone / Mobile a montré la fin du modèle Wintel et de l'hégémonie de Windows face aux ogres Android et iOS. Le rachat de Nokia n'a pas renversé la situation. Windows reste le système écrasant sur PC mais est-ce encore l'essentiel ? A court terme oui sans doute. Mais sur le long terme, non. Attention Microsoft ne va arrêter Windows demain. Non, mais il ne représente plus le coeur de l'activité et sans doute qu'en interne, les équipes systèmes pèseront moins. Un changement de culture.
Mais il ne faudrait pas que Microsoft délaisse certains marchés prometteurs. Prenons la réalité virtuelle / augmentée, Windows Mixed Reality offrait un potentiel même si le marché tarde à réellement décoller. Plusieurs constructeurs supportent la plateforme (Acer, Dell, HP, Lenovo) mais elle tarde à se faire une réelle place. Le store dédié crie famine. Et c'est un serpent qui se mord la queue : les utilisateurs veulent du contenu et les éditeurs / développeurs veulent des utilisateurs…

Java : procès Oracle – Google, manche Oracle !



L'affaire Oracle – Google nous suit depuis 8 ans et ce n'est pas fini. Petit rappel : Oracle, qui a racheté Sun et donc Java, accuse Google d'utiliser des API pour développer Android alors qu'elles sont protégées et donc soumises à royalties. Bref, Google doit passer à la caisse… Au printemps 2016, une décision affirme que ces API ne sont pas soumises à licence et rentrent dans le cadre du "fair use". Et tout tourne sur cet usage raisonnable des 11 000 lignes de code. Pour Google, l'usage a toujours été raisonnable, la loi américaine le permettant. Oui ces 11 000 lignes étaient indispensables pour développer Android mais comme le système est gratuit, il ne génère pas de revenus. L'éditeur veut bien payer mais pas les 8,8 milliards $ demandés par Oracle.

Oracle élargit alors sa plainte, et demande un nouveau procès, en prétextant que Google préparait les apps Android à fonctionner sur ChromeOS en cachant le projet ARC++. Demande rejetée. Mars 2018 : c'est un peu à la surprise générale qu'une nouvelle décision relance notre affaire. Le tribunal estime que Google ne fait pas un usage raisonnable des API. Cette victoire d'Oracle n'est pas décisive. Google peut saisir la cour suprême des Etats-Unis, la plus haute juridiction américaine.

L'affaire est loin d'être terminée. Il faudra encore plusieurs années avant que le jugement définitif ne soit prononcé. Si Google perd, restera à savoir le montant à verser à Oracle. C'est un des enjeux : si les juges confirment l'exigence d'Oracle, c'est tout l'écosystème Android qui va être ébranlé. Si le montant ne dépasse pas 1 milliard $, la défaite ne sera pas aussi dure qu'il n'y paraît.

Si Oracle perd, la question de l'avenir de Java chez l'éditeur se posera, une nouvelle fois. Mais en un an, l'éditeur a confié des morceaux entiers du monde Java à l'extérieur. L'exemple le plus marquant est J2EE qui a rejoint Eclipse.

COMMUNAUTÉS

Annoncez vos meetups, et conférences sur Programmez ! :
ftonic@programmez.com

GDG Toulouse
17 mai : WebVR et intégration continue.

ParisJUG
15 mai : spécial 10 ans de ParisJUG !

MAI

PHP Tour 2018
PHP Tour 2018 s'arrêtera à Montpellier les 17 et 18 mai. Le programme s'annonce chargé, comme toujours : moteur PHP 7, RGPD, tests, Rex, Synfony, le rôle de la documentation, l'asynchronisme, OpenAPI, GraphQL, etc.
Programme complet :
https://event.afup.org/phptourmontpellier2018/programme/

JUIN

AI Paris 2018
Les 11 et 12 juin 2018 ouvriront les portes du rendez-vous le plus attendu de la scène AI hexagonale : AI Paris 2018 !
Pour cette deuxième édition le congrès invitera l'ensemble des décideurs et utilisateurs métiers pour 48h d'immersion dans l'ère du business augmenté !
Pour en savoir plus :
https://tinyurl.com/y9ko5w9x

EclipseCon 2018
Les 13 et 14 juin, la grande conférence EclipseCon se tiendra une nouvelle fois à Toulouse.
https://www.eclipsecon.org/france2018/

DevFest Lille
Le Devfest Lille 2018 se passera le 21 juin 2018, 400 personnes prévues. La conférence prendra place dans les locaux de l'IMT Lille-Douai,
20 Rue Guglielmo Marconi,
59650 Villeneuve-d'Ascq). Le site est accessible via
https://devfest.gdglille.org/

Hack in Paris
Du 25 au 29 juin, Hack in Paris sera la grande conférence sur la sécurité et le hacking en France. Un événement à ne pas râter !

CONFÉRENCES TECHNIQUES

Xebia organise 4 conférences techniques
- DataXDay, conférence autour du futur de la Data Science, des prochaines innovations en matière de temps réel et des incontournables en architecture Big Data. 17 mai 2018 - dataxday.fr
- Paris Container Day, conférence pionnière en France dédiée à l'écosystème des conteneurs. Cette année, le thème sera : "Vivre avec l'Orchestration". 26 juin 2018 - paris-container-day.fr/
- FrenchKit, la première conférence française dédiée aux développeurs iOS et macOS. 20 et 21 septembre 2018 - frenchkit.fr/
- XebiCon, la conférence qui vous donnera les clés pour tirer le meilleur des dernières technologies : Data, Architecture, mobilité, DevOps, etc. 20 novembre 2018 - xebicon.fr

À VENIR

Kansas Fest (USA)
Kansas Fest 2018 se tiendra à Kansas City du 17 au 22 juillet. Il s'agit de la 30e édition de cette conférence réunissant les fervents de l'Apple II venant du monde entier. La session plénière sera animée par Roger Wagner rendu célèbre par sa série d'articles Assembly Lines.
https://www.kansasfest.org/
https://www.programmez.com/actualites/40-ans-la-passion-de-lapple-ii-ne-faiblit-pas-26333

Apple II Festival France
La quatrième édition du petit frère français de Kansas Fest aura lieu au 1er au 5 août 2018 au Maska à Castéra-Verduzan dans le Gers. Au programme ici aussi des sessions sur le matériel et le logiciel, des ateliers et beaucoup de convivialité.
https://www.apple2festivalfrance.fr/

TOUT EST INCLUS
dans un environnement totalement intégré, en français

- Environnement de Développement Professionnel
- Tout est en **français** (manuels, logiciel, exemples)
- Crée des **.EXE** sécurisés, des **Webservices**, des applications **.NET** et **Linux**, des applications **Java** (Windows et Linux)...
- Code **multi-plateformes** recompilable en natif sur **Mobile** (iOS, Android, CE) et **Internet** (Intranet, Extranet, SaaS, Cloud...)
- **Cloud, SaaS**
- Gestion de la **RGPD**
- **Tous les Windows**: 10, 8, 7, Vista, XP,...
- Générateur automatique d'**IHM (UI/UX)**, avec charte graphique automatique. Création automatique de superbes fenêtres par utilisation de gabarits fournis
- Champs bureautiques inclus pour vos applications: Tableur, Traitement de Texte, Lecteur PDF...
- Générateur de **Rapports et de Requêtes diffusable gratuitement**, création de **PDF**, codes-barres, étiquettes. Fond de page PDF
- **Menu d'export automatique** dans vos applications : vers Word, Excel, OpenOffice, XML, PDF; Graphiques 3D; Historique de saisie,... Envoi d'email, Macros
- **RAD** : Générateur d'applications complètes, avec possibilité de créer ses propres Patterns
- **HFSQL**, Base de Données incluse: Client/Serveur, Locale et Mobile sous Windows, Linux, Android, iOS (**libre** et gratuite); Gère 4 millions de Téra-octets; cluster, **Cloud**
- Accès aux Bases de Données tierces: **Oracle, AS/400, SQL Server**, DB2, **MySQL**, PostgreSQL, Informix, Access, xBase, SQLite, etc... BigData
- **Réplication** entre bases de données
- **XML** natif
- **Webservices** REST et SOAP (XML, JSON,...)
- Accès natif à **SAP R/3**, Lotus **Notes**, **Google** Agenda, Earth, Map, Contact,... et SalesForce, LDAP, Outlook,...
- Centre de Modélisation **UML**, Merise et autres; code généré depuis l'analyse, reverse engineering
- Centre de suivi du **planning** d'équipes
- **Télémétrie** pour vos applications
- Tableau de Bord de **suivi de projets**
- **Dossier** automatique : analyse & programmation
- **Audit** dynamique et statique
- Règles métier; **Intégration continue**
- Création et utilisation de **composants** ; 3-tier
- Centre de suivi des retours utilisateurs
- Langage de 5° Génération **L5G**, élimine 90% du code. Programmation Linéaire ou POO, MVP
- **Ouverture** à C++, C#, Java, VB, Cobol...
- Fonctions **Domotique** et **IoT**
- Gestion **liaison série** RS 232, parallèle et **USB**
- Fonctions Bluetooth
- Fonctions réseau **SNMP**
- Fonctions TAPI, OPC, FTP, HTTP, Socket, Twain, API, DLL, Blockchain, oAuth, ...
- Fonctions Multimédia (image, son, vidéo)
- Editeur de code intelligent, avec test immédiat sans recompilation
- Gestion des **versions** (en local, à distance)
- Automate de **tests unitaires** de code et d'IHM, Editeur visuel de tests de **non-régression**
- Débogueur puissant: threads, composants,... **Débogage à distance**
- **Profiler**, pour optimiser la vitesse du code
- **Build** programmable
- **Robot** de surveillance et monitoring
- Multilingue automatique: jusqu'à 64 langues
- Générateur d'**Installations en 1 clic,** «Live Update», Install-push, silencieuse
- Auto-formation facile, en 1 semaine

*Abonnés: Retrouvez le documentation WINDEV **(108 pages)** jointe à ce numéro*

4 développeurs sur 10 pensent changer de métier dans les 5 ans

Panique ! Est-ce que le métier de développeur n'intéresse plus ? Trop de pression ? Une étude menée par Le Blog du Modérateur et Hellowork.io fournit des éléments de réponse. Cette enquête a été menée sur 836 développeurs, salariés et freelances.

Comme pour Flavien Chantrel (Blog du Modérateur), ce chiffre m'a interpelé. Comment l'interpréter, comment le comprendre ? Comme nous l'a précisé Flavien, il y a clairement une envie d'évolution, d'avoir un meilleur salaire ainsi qu'une projection sur un périmètre plus large. Ce qui s'en dégage c'est l'idée de ne pas être développeur 10-20 ans et d'aller vers des postes de management, de chef de projets, bref de voir autre chose.

N'est-ce pas une impression de déjà vu ? Il y a 10-15 ans, nous entendions la même chose de la part des étudiants : pas envie d'être développeur toute ma vie, aller vers un profil de chef de projet…

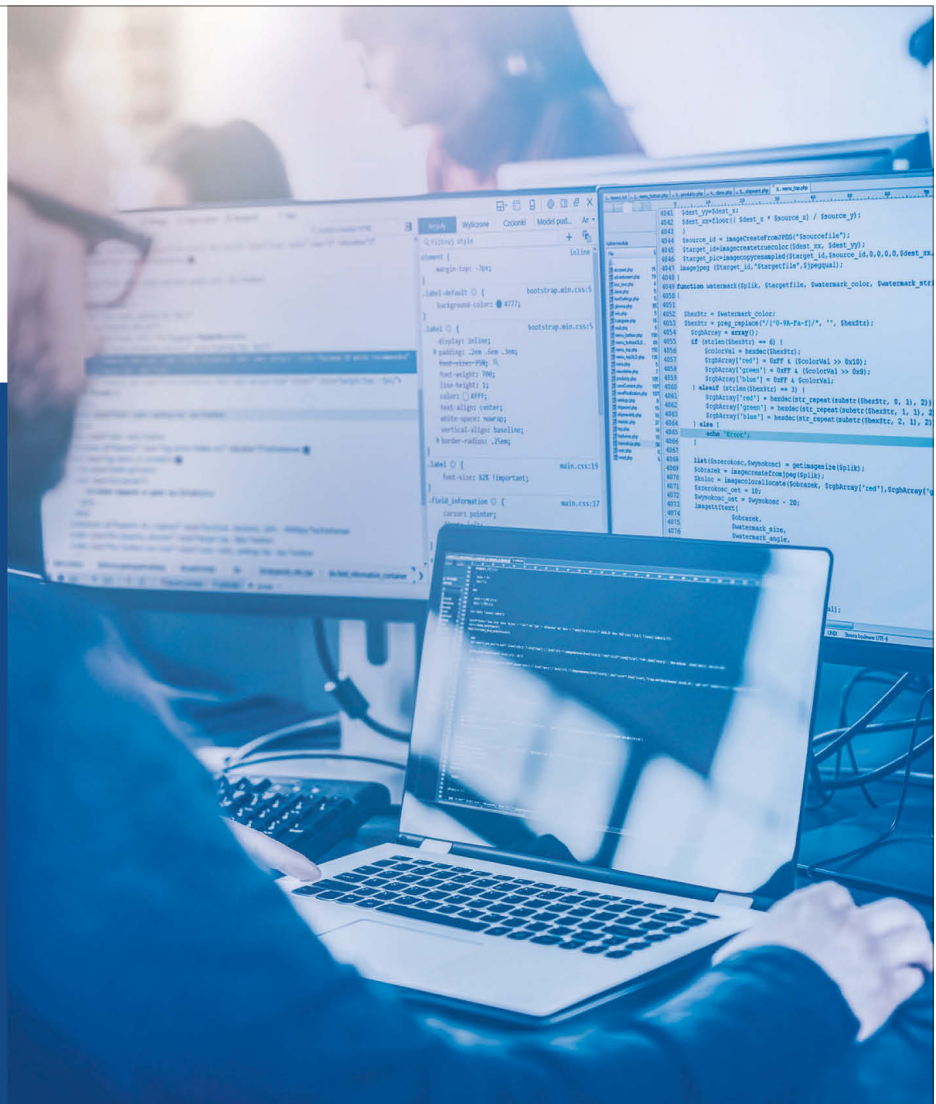
Cependant, pour Flavien, il y a ce que l'on pense et la réalité de la vie. L'envie de changer de métier, de se reconvertir sont des réflexes qui ressortent régulièrement dans les enquêtes. "C'est le besoin d'avoir une soupape, de s'ouvrir d'autres portes." Précise Flavien. Comme avec le ressenti en météo, il y a le ressenti immédiat par rapport à son job du moment. Et ce n'est pas pour autant que 84 % des développeurs vont changer de métier. Elément intéressant, comme le pointe l'étude, on ne rejoint plus une entreprise pour 15-20 ans, mais plus pour un projet qui va durer 2-3 ans. Ceci pour ensuite se donner la possibilité de changer, de chercher un autre défi. Mais globalement, le développeur aime son métier (environ 2 sur 3). Le salaire devient un critère majeur (75 %). Cela s'explique par plusieurs éléments factuels, liste non exhaustive :

- Les développeurs en poste sont sollicités : c'est une réalité à ne pas négliger par les entreprises. Les développeurs, selon les profils bien entendu, reçoivent régulièrement des propositions
- Le fait d'entendre partout "on cherche des développeurs, il y a une pénurie de développeurs, etc." facilite l'idée que je peux aller ailleurs, avoir un meilleur salaire.
- L'environnement, les technologies, les projets sont d'autres facteurs importants

"TROUVER UN ÉQUILIBRE ENTRE VIE PRO ET VIE PRIVÉE"

Les freelances représentent seulement 11 % des répondants. Ce qui est finalement peu. Et surtout, le freelance n'est pas un profil excitant. "Beaucoup se projettent dans une entreprise, TPE, PME, grandes entreprises, mais seulement 1 sur 10 se projette comme freelance" analyse Flavien. Il semble que les aspects négatifs l'emportent : comptabilité, la pression, trouver des clients, etc.

•



Openska

Formations et coaching

Web, Data et DevOps

"être un meilleur Tech Leader"

www.openska.com

Baromètre HIRED recherche d'emploi

Les technologies les plus en vogue qui peuvent faire la différence sur votre C.V.

Hired.com (Hired), la plateforme de recrutement spécialisée dans les métiers de la technologie publie ce mois-ci en exclusivité sur Programmez! le baromètre des technologies les plus recherchées par les entreprises aujourd'hui qui peuvent permettre aux développeurs en recherche d'emploi de se démarquer de leurs concurrents et d'obtenir plus facilement des entretiens.

« *Avec l'accélération de la transformation numérique et le rôle croissant des développeurs, les besoins des entreprises sur les technologies actuelles évoluent plus rapidement qu'on ne le pense. Aujourd'hui, il est donc essentiel pour les développeurs de s'assurer de suivre les tendances et d'apprendre les nouveaux langages ou frameworks qui peuvent les aider à se démarquer et à obtenir plus facilement des entretiens.* » explique Antoine Garnier-Castellane, Directeur du bureau français de Hired. Chaque mois, l'équipe de Hired partagera sur Programmez! en exclusivité les évolutions et les tendances qu'elle observe sur sa plateforme de recrutement qui peuvent impacter la carrière professionnelle des développeurs aujourd'hui. On remarque pour le mois de mars et les six derniers mois les tendances suivantes :

- Java est sur les six derniers mois (Octobre 2017 à Mars 2018) la technologie que l'on retrouve le plus dans les candidatures postées sur Hired avec 17,08 %. Et pourtant la demande des entreprises pour la technologie Java est en baisse et Java n'est qu'à la 10ème place des technologies qui suscitent le plus de demandes d'entretiens pour les candidats.
- React, GO et Node.js sont les trois technologies qui sont le plus en demande depuis les six derniers mois. Les ingénieurs qui maîtrisent ces technologies suscitent de plus en plus de propositions d'entretiens de la part des entreprises sur la plateforme.
- Pendant la période d'octobre 2017 à mars 2018, Node.js est la seule technologie à se retrouver dans le peloton de tête avec 14,3 % de candidatures pour les développeurs et un nombre moyen d'entretiens s'élevant à près de 10 par candidats.

A propos de HIRED

HIRED.com est une plateforme technologique de recrutement qui attire et sélectionne les meilleurs profils techniques du marché afin de permettre aux entreprises de recruter des candidats qualifiés rapidement et efficacement. Chaque semaine, entre 50 et 100 nouveaux candidats de la communauté française HIRED en recherche active (développeurs, data scientists, designers, etc.) sont triés sur le volet grâce à des algorithmes et prêts à être contactés. Start-up d'origine américaine, HIRED a levé plus de 100 millions de dollars en 4 ans.

OCTOBRE 2017 - MARS 2018

Technologies demandées	Pourcentage de candidatures de développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	17,08%	React	11,48
PHP	14,69%	Go	11,00
Node	14,03%	Node	9,80
Python	11,83%	DevOps	9,57
React	11,07%	Vue	8,52
Angular	9,92%	Angular	8,45
Android	5,06%	Ruby	7,98
Ruby	4,58%	Python	7,65
.NET	3,91%	PHP	6,81
Go	2,96%	Java	6,12
iOS	2,67%	.NET	5,88
Vue	2,19%	Android	5,68
DevOps	1,34%	iOS	5,57

MARS 2018

Technologies demandées	Pourcentage de candidatures de développeurs	Technologies demandées	Nombre moyen de demandes d'entretiens
Java	15,69%	Go	9,75
PHP	15,33%	React	9,37
Python	12,77%	Node	7,34
Node	12,77%	Python	7,26
React	10,95%	PHP	6,81
Angular	9,85%	Angular	6,52
Android	5,11%	Ruby	6,00
.NET	4,74%	Vue	6,00
Ruby	4,01%	Android	5,14
iOS	3,28%	.NET	5,08
Go	2,92%	Java	4,91
Vue	2,55%	DevOps	4,60
DevOps	1,82%	iOS	4,22

François Tonic

AltairDuino : un Altair comme en 1975 :-)

Retour 43 ans en arrière. Nous sommes en 1975, un constructeur spécialisé dans l'électronique, MITS, décide de se lancer dans le micro-ordinateur. Le micro-ordinateur était déjà connu en France grâce au Micral N, le tout premier ordinateur au monde. La révolution est possible avec l'utilisation du microprocesseur, le fameux CPU : l'Intel 8080. MITS est dans la brèche ouverte par le Micral et sort l'Altair 8800 !

Il est lourd et gros. Il utilise le processeur 8080A d'Intel cadencé à 2 Mhz. Il embarque une mémoire vive de 256 octets, extensible à 64 Ko ! L'Altair bouleverse l'informatique de cette époque : il tient dans un boîtier "compact", il est extensible (par des cartes) et son prix de vente est très raisonnable. Deux versions sont proposées : en kit à assembler ou tout monté. Le succès arrive immédiatement. Et pourtant, l'Altair n'a ni clavier, ni écran, ni stockage. Les commandes se font en actionnant des commutateurs en façade et les réponses sont données par des LEDS. Il est possible de lire des programmes grâce à des bandes perforées, en rajoutant un lecteur de bandes… K7 et disquettes arriveront après. Deux inconnus, Paul Allen et Bill Gates, vont créer le premier BASIC accessible : l'Altair Basic. Eh oui, c'est grâce à l'Altair 8800 que Microsoft est né ! Cette histoire est un des mythes fondateurs de la micro-informatique : quand les deux développeurs obtiennent un rendez-vous, le BASIC n'était pas développé. Un mois plus tard, le langage est prêt à être testé alors qu'il avait été créé sur un autre ordinateur car ils ne possédaient pas d'Altair. La démonstration se déroule très bien…

Aujourd'hui, l'Altair est très difficile à trouver. Et prévoyez un solide compte en banque pour vous l'offrir : 2500 à 3000 minimum.

Bonjour l'AltairDuino !

Ou sinon, vous pouvez opter pour une solution plus simple : l'AltairDuino. Comme son nom l'indique, il s'agit d'une réplique fonctionnelle de l'Altair8800 basée sur une Arduino Uno avec un processeur ARM 32 bits et 512 Ko de RAM. Nous retrouvons le même principe de commutateurs et de LEDS…

Deux modèles sont possibles : en kit à monter et à souder soi-même ou entièrement monté. Le montage n'est pas forcément difficile mais il est long et il faut aimer la soudure. Comme ce n'est pas notre grande passion, nous avons opté pour la version montée.

Au déballage, nous savons qu'il ne s'agit pas du vrai de 1975 mais nous sommes impressionnés par la bête. C'est une expérience à part.

Pour faciliter son utilisateur et surtout pouvoir utiliser les programmes disponibles sur la machine, on se connecte à l'AltairDuino en USB, Bluetooth ou en série. Pour fonctionner, un simulateur d'Altair a été développé (Altair 8800 Simulator) est embarqué sur l'Arduino.

Ce simulateur offre la capacité de mémoire de l'original (64 Ko) et il tourne à la même vitesse. Il supporte les lecteurs de disquettes et les disques durs de l'Altair. Plusieurs logiciels sont disponibles par défaut sur notre clone : l'Altair Basic, CP/M, Zork, Altair DOS, Supercalc. De quoi déjà tester la bête.

Putty & commutateurs

L'AltairDuino est livré avec un câble USB et son alimentation. Allumez la machine.

PUTTY est vivement recommandé ou un équivalent si vous n'êtes pas sous Windows. Il faut repérer le port de communication de la carte Arduino puis dans PUTTY mettre le bon port COM et la vitesse 115200 bauds. Si la connexion est faite, les LEDS clignotent brièvement. Et la session PUTTY s'ouvre.

Comme nous sommes sur un Altair, nous allons jouer du commutateur. Chargeons maintenant l'ALTAIR BASIC :

- on lève les commutateurs 1 et 12;
- on abaisse AUX2;
- sur le Terminal : DISK02.DSK Altair DOS 1.0 doit s'afficher;
- on abaisse les commutateurs 1 et 12 et lève le 3;
- on baisse AUX1;
- on boote sur le disque ;
- Si tout se passe bien, le terminal affiche : MEMORY SIZE. On appuie simplement sur Entrée;
- INTERRUPTS : on tape Y;
- HIGHEST DISK NUMBER : on tape 0;
- DISK FILES et RANDOM FILES : on répond 4.

Si tout se passe bien le terminal affichera :

055797 BYTES AVAILABLE

DOS MONITOR VER 1.0

Pour réinitialiser l'Altair, il suffit de : lever STOP puis RESET. La machine est prête à charger un autre programme.

Notez que l'ALTAIR n'aime pas les minuscules.

Conclusion

Utiliser l'AltairDuino c'est comme retourner dans l'âge de pierre de l'informatique. Mais honnêtement, c'est une expérience à faire! Surtout quand on joue à actionner les commutateurs. Rassurez-vous, on trouve rapidement les réflexes… Pour les curieuses / ieux, nous vous invitons à lire la documentation : Altair 8800 Simulator.

Un grand bravo à l'équipe d'AltairDuino ! Et nous, on l'adore notre Altair. •

Tarifs :

modèle en kit : 149,95 $

modèle assemblé : 249,95 $

https://www.altairduino.com

Les +

- **Ambiance 1975**
- **Qualité du modèle**
- **Les logiciels**
- **Interface Putty**

Les –

- **Logiciels disponibles limités**
- **Manque d'exemples pour aller plus loin**

François Tonic

1998-2018

Partie 1

20 ans de technologies dans Programmez!

Difficile pour un étudiant ou même un jeune développeur d'imaginer les technologies, les outils et les langages que nous avions à notre disposition en 1998. Cela pourrait être la préhistoire de l'informatique, un monde fait d'ordinateurs dinosaures, de la programmation avec de gros claviers, le téléphone portable, bah un téléphone portable, Internet par un simple modem. Limite si l'on n'utilisait pas encore le boulier. Linux était à peine né, Apple au bord du gouffre, Microsoft plus puissant que jamais, on ne parlait pas de Google, Facebook, Twitter, etc. Netscape régnait en maître sur le monde des navigateurs. Les disquettes étaient toujours utilisées et on achetait les logiciels en magasins. La préhistoire je vous dis :-) La rédaction

N° 1 mai-juin 1998 : Java

Java avait à peine 3 ans et il était déjà un peu partout. On se posait la question de savoir si une JVM allait être installée par défaut dans IE 5 (le 5 même pas encore le 6). Après ce qui est amusant, c'est de lire la conviction de certains gourous. « Delphi peut coiffer le C++ dans un contexte précis ». On connaît la suite et les difficultés de Borland, tout comme la chute de Delphi, avant son retour récent auprès des développeurs. Mais nous avions un œil sur l'Open Source, le logiciel libre et Linux. Et il est intéressant de voir Richard Stallman parler de Free Hardware. Mais c'est l'occasion aussi de se souvenir de l'effervescence autour des IDE. Si aujourd'hui, la plupart des développeurs utilisent peu ou prou les mêmes (Visual Studio, Eclipse, IdealJ, Xcode pour les principaux), à la fin des années 90, l'offre était bien plus large qu'aujourd'hui. Rappelez-vous de Visual J++, de Visual Age, Jbuilder, OpenJ ou encore Visual Café ou encore de Visual Basic. La grande époque !

Ce qui m'avait marqué immédiatement, c'était la couverture avec la fameuse bague Java de Sun : la Java Ring. Un des flops du monde Java.

Flash : de la gloire à la descente aux enfers

Ah Flash ! Nous en avons écrit et publié des articles et des dossiers dans Programmez!. Flash fut une des technologies web les plus utilisées par les vidéos et les animateurs durant une quinzaine d'années ! Rappelons qu'il s'agissait de l'environnement le plus connu sur la partie auteur et la partie runtime. On disposait d'un environnement de conception complet, d'un langage et d'une portabilité sur les navigateurs web.

Microsoft avait tenté de concurrencer avec Silverlight. Prometteuse et plutôt bien outillée, cette technologie n'a jamais réussi à trouver une position dominante.

Mais Flash va subir une double offensive qui lui sera fatale : HTML 5 et l'explosion du web mobile grâce aux smartphones. Rappelez-vous de la tribune de Steve Jobs en 2010 contre Flash. Il fallait sortir Flash des navigateurs, imposer HTML 5 et de nouveaux codecs non-propriétaires. Le fait que l'iPhone ne proposait pas de support de Flash a fait très mal. Puis, les navigateurs et les services web ont retiré les plug-ins et le support du format d'Adobe. Adobe annonce la mort définitive de Flash pour 2020. Mais finalement, c'est déjà le cas.

Le Devsigner : un profil trop chargé

C'était il y a 10 ans. Adobe, puis Microsoft poussaient un nouveau profil à quatre mains : le Devsigner, combinaison de développeur et de designer web. L'idée était de réunir les deux compétences dans un unique profil pour faciliter les développements web. Ces deux profils discutaient mal ensemble, ce qui n'a pas forcément beaucoup changé…

"Le devsigner est né en grande partie par l'évolution des outils, des technologies. L'évolution du processus de production web va totalement dans ce sens. Désormais, on retrouve un cycle de développement incluant outils et profils différents. Cette évolution nécessite de bonnes pratiques, de nouveaux réflexes. Il faudra, pour le designer, acquérir une rigueur quand il crée une interface avec des objets d'interface. Il doit absolument nommer chaque objet, chaque élément. Nommage vital pour le développeur… Bref, le designer, même s'il demeure loin du code pur, touche à un élément sensible. Nécessitant une rigueur des process de développements et de bonnes pratiques. " Programmez! 108, mai 2008.

Chaque éditeur avait imaginé des suites d'outils intégrés (plus ou moins bien) pour fluidifier les projets, la gestion des projets,

centraliser le code et le design. Chez Microsoft, nous avions l'excellente gamme Expression dont l'outil Blend.

Nous étions dans la mouvance Flash, Silverligh, des applications RIA (Rich Interface Application). Adobe proposait bien entendu sa suite Creative et surtout une plateforme de développement, Flex.

Pour Adobe, il s'agit d'aller vers le marché des développeurs qui était peu connu d'eux et Microsoft voulait parler au designer, un marché là aussi peu connu d'eux. Mais le devsigner tomba rapidement dans la fosse des fausses bonnes idées.

Résultat : Adobe lâcha peu à peu les outils de développement et Microsoft fit de même avec la plupart des outils Expression. Malgré tout, dix ans après, on reparle de cette collaboration indispensable entre développeur et designer notamment avec les technologies de réalités virtuelles / augmentées / mixtes, les IoT car il faut repenser l'ergonomie et l'interface.

Ruby

Le langage Ruby, apparu en 1995, a été popularisé avec son framework Rails (Ruby on Rails). Il demande un effort d'apprentissage pour maîtriser la syntaxe et son fonctionnement, mais il a l'avantage d'être un langage facile à lire. Il a connu une certaine notoriété, mais depuis plusieurs années, Ruby s'est fait bien plus discret ainsi que Rails même si ces profils sont toujours recherchés.

Mais l'arrivée de nouveaux langages et la confirmation d'autres, tels que Rust ou Python, ont fait mal à Ruby. Et Rails, malgré d'évidentes qualités reste confidentiel en usage.

Internet Explorer et Edge

Internet Explorer est une longue histoire qui a connu bien des obstacles. IE6 a laissé de très mauvais souvenirs aux développeurs web. Il a connu une longue vie grâce à Windows XP. Respect des standards aléatoires, contorsions pour avoir un affichage correct et surtout le monopole des navigateurs web. Mais IE6 accumulait surtout les failles de sécurité et les bugs. Et Microsoft mit de longues années avant de proposer un successeur, IE7, corrigeant les gros défauts. Et cette obstination a profité à Firefox et Chrome qui finirent par dépasser Internet Explorer. Et c'est cette mauvaise tendance qui a fait bouger Microsoft. Les versions suivantes ont été plus respectueuses des standards et des évolutions des API web, ce que IE6 ne faisait pas…

Mais le mal était déjà fait et malgré d'importants progrès, Microsoft s'est retrouvé dans une impasse. Firefox prit le marché à son compte, mais c'est finalement Chrome de Google qui remporte la mise, notamment grâce au moteur WebKit et à son fork.

Edge a été annoncé comme le big bang du navigateur par Microsoft : nouveaux moteurs, nouvelle architecture. Mais malgré tout, ce nouveau navigateur, qui a lui aussi cumulé les défauts de jeunesses et des mises à jour aléatoires n'arrive pas à se faire une place sur le marché des navigateurs. En mars 2018, Edge stagne à 4,5 %, IE pointe à 12 %, Chrome dépasse les 61 % ! (Chiffre netmarketshare).

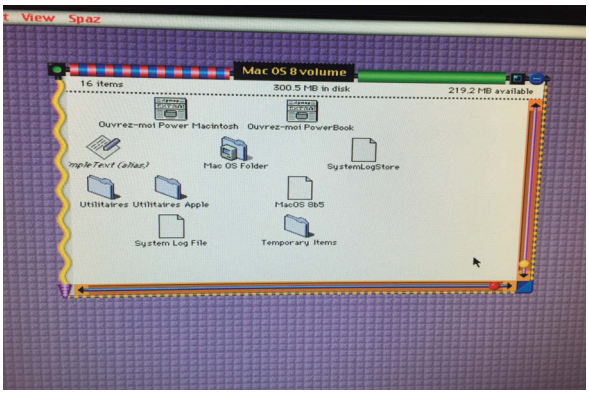
Sur mobile c'est encore pire. Le navigateur Microsoft est inexistant. L'échec de Windows Mobile y est pour beaucoup. La disponibilité de Edge sur Android et iOS peut-elle changer la situation ? Nous n'y croyons pas.

La danse funeste des systèmes

L'échec Copland

Depuis les premiers micro-ordinateurs au milieu des années 1970, les systèmes n'ont cessé d'évoluer : ils naissent, ils grandissent et ils meurent. Et de nombreux OS ont déjà disparu en 1998 : CP/M, QDOS, ProDOS, GEM, GEOS, NeXTStep, BeOS (quasiment mort)… Windows est l'OS dominant depuis la sortie des versions 3.0 et 3.11. Sur la partie serveur, Unix, et désormais Linux, reste présent. OS/2, initialement développé par IBM et Microsoft, sombre peu à peu. MSX, le standard est déjà un souvenir.

Apple a trouvé le successeur de son Mac OS, descendant en droite ligne du premier System, développé en 1983-84 et arrivé à bout de souffle. La Pomme avait fourni un énorme effort sur le projet Copland. Copland était une réécriture complète de l'OS, natif aux processeurs PowerPC et intégrant tous les mécanismes modernes (multitâche préemptif, multiprocesseur, interface nouvelle, piles réseaux les plus récentes, micro-noyau, architecture modulaire). Copland est un projet tentaculaire mobilisant plusieurs centaines de développeurs et engloutissant +500 millions $ alors que le constructeur est au bord de la faillite. Le développement n'est pas assez intégré et le projet devient ingérable. Une première version bêta est attendue pour début 1996, mais rien n'arrive. Les versions delta sont peu fiables et nécessitent deux Mac pour pouvoir booter correctement. Et devant l'impossibilité de stabiliser les nombreuses briques techniques, en août 1996, le projet est arrêté net ! Copland ne sera jamais le successeur du vénérable Système ! Il faut chercher en urgence un OS extérieur. Et la pression est énorme : Windows 95, avec son système plug&play et sa nouvelle interface, impose définitivement Microsoft.



Copland

BeOS est un système très en vue à ce moment-là. En développement depuis 91, les versions se sont succédées mais son éditeur, Be Inc., échoue à en faire une alternative crédible au Mac et à Windows. Même si BeOS évolue régulièrement et stabilise son code, il a bien du mal à sortir en version de production. Et il manque cruellement d'applications ! Eh oui, les applications sont déjà le nerf de la guerre ! BeOS sort une version PowerPC pour séduire Apple, puis une version Intel, pour élargir son public. Programmez! proposera même la version R5 sur son CD-ROM mensuel !

Dans un premier temps envisagé, BeOS est écarté, pour des raisons techniques et financières. Cet échec signe la mort programmée du système qui sera vendu et plusieurs « clones » reprendront l'esprit BeOS : Zeta et surtout Haiku, qui est toujours en développement. La première version bêta est attendue pour 2018, la dernière Alpha remonte à 2012… Apple porte son choix sur NeXTStep / OpenStep de NeXT, la société de Steve Jobs. C'est le retour du cofondateur ! En quelques mois, les équipes repackagent le système sous le nom de Rhapsody. Rhapsody n'a jamais été

une version utilisateur, mais il servira de fondations au futur Mac OS X dont la première bêta officielle sort en l'an 2000.
Apple, en attendant la transition vers ce nouvel OS, gère l'existant avec un MacOS 8, puis 9, intégrant de nombreuses technologies initialement prévues dans Copland. Mais faute de fondations techniques nouvelles, les avancées sont moins radicales. Et surtout, Apple va intégrer à Mac OS X une compatibilité avec les applications MacOS classique, un travail essentiel pour faciliter la migration et laisser le temps aux développeurs.

Windows Longhorn

Les mauvaises versions, pour ne pas dire désastreuses versions, de Windows s'enchaînent. La plus mémorable est Windows Millenium. En 2001, Microsoft sort Windows XP. Cette version s'impose rapidement. Elle efface les errements de la lignée Windows 95 et introduit une interface plus efficace et des fondations plus récentes. Dès cette date, les équipes développent un tout nouveau système particulièrement ambitieux : Windows Longhorn.
Le projet est particulièrement ambitieux avec des nouveautés dans toutes les couches. Trois couches vont particulièrement être mises en avant : WinFS doit être le nouveau système de fichiers s'appuyant sur XML et le moteur de base de données Yukon, WinFX est le nouveau modèle de développement (qui sera renommé en .Net 3.0) et l'interface. L'interface graphique a changé plusieurs fois durant le développement de Longhorn, ne facilitant pas le travail des développeurs. L'interface Aero Glass deviendra l'interface de Windows à partir de Vista. Il s'agit d'une évolution majeure par rapport XP.
Le projet Longhorn va être « rebooté » en été 2004 pour pouvoir sortir une version stabilisée en 2006 : Windows Vista. Pour ce faire, le projet subit des coupes dans les technologies et particulièrement WinFS.

Longhorn

« Microsoft a annoncé, contre toute attente, une modification de sa stratégie sur plusieurs autres technologies clés de Longhorn. La couche de développement WinFX (le nouveau modèle de programmation) sera disponible pour Windows XP et Server 2003 ! Cette annonce est particulièrement importante. Des voix s'élevaient contre le modèle WinFX apparaissant uniquement dans Longhorn et donc d'une incompatibilité des applications WinFX avec les anciens OS. » Programmez! n° 68, octobre 2004

Skeuomorphisme ou comment rendre illisible une interface

L'interface doit-elle être réaliste ou abstraite ? Doit-elle avoir du relief ou être plate ? Les premières versions d'iOS avaient imposé une idée précise de l'interface utilisateur en usant et en abusant du skeuomorphisme que l'on voyait aussi ailleurs.
« Le skeuomorphisme définit une tendance visuelle et ergonomie des objets composant une interface. Par définition, le skeuomorphisme ne définit pas un design selon sa fonction, mais selon l'objet d'origine qu'il évoque. Par exemple, reprendre l'aspect cuir d'un portefeuille ou d'un agenda, un tableau blanc, le câble réel pour représenter un câble, etc. » Programmez! n° 156, octobre 2012. Finalement, cette tendance ultra réaliste a été supplantée par la tendance inverse, le « plat design » : on aplatit l'interface et on simplifie le design des objets. Windows Phone a été le premier à l'imposer, avec une certaine réussite. Puis iOS et Android ont suivi et l'ont imposé, parfois avec plusieurs itérations pour affiner et rendre lisibles certaines parties de ces interfaces plates.

Le projet Mono se transforma en Xamarin

Nous nous souvenons encore des conférences des équipes Mono menées par Miguel de Icaza en dehors des conférences européennes TechED, car elles n'étaient pas les bienvenues. Et pourtant le projet Mono ne fut pas de tout repos. L'idée de départ est simple : créer une implémentation open source du framework .Net de Microsoft, en suivant les modules ouverts par Microsoft et disponibles dans les standards ECMA. Il supporte le langage C#, le compilateur, le runtime CLI et les librairies. Alors que Microsoft tardait à porter .Net sur d'autres systèmes que Windows, Mono voulait être multiplateforme. Les premières versions, été 2001, se résumaient à peu de choses : compilateur, runtime. Il aura fallu plus de 3 ans de travail pour sortir Mono 1.0 avec l'ECMA CLI, le compilateur C# 1.0, le runtime, l'interface, la toolchain, les librairies. Un IDE compatible était déjà proposé : MonoDevelop. Ce travail se fait au sein de la société Ximian, rachetée par Novell en 2003.
Le rachat de Novell par Attachmate pèse sur Mono qui n'est pas une priorité pour le nouveau propriétaire. En 2011, Xamarin naît ! Les équipes décident de miser sur le développement mobile, toujours basé sur Mono. Peu à peu, Xamarin devient une des plateformes de référence pour les développeurs mobiles.
Et en mars 2016, Microsoft annonce le rachat pur et simple de Xamarin qui est intégré dans Visual Studio. Quel retournement de l'Histoire !

Windows Phone / Windows Mobile : un bon système ne suffit pas !

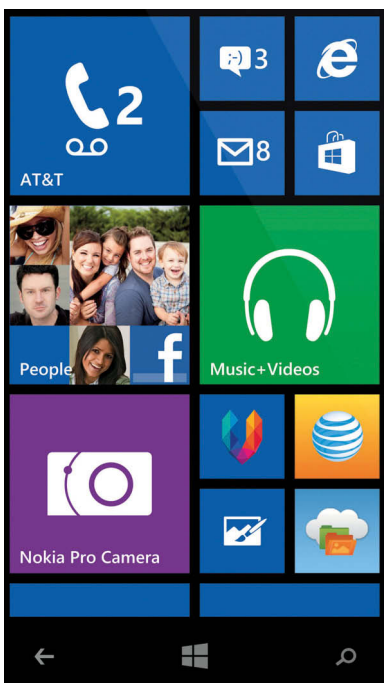
Les relations de Microsoft avec la mobilité n'ont jamais été simples. Souvenez-vous, peu après la sortie du premier iPhone en 2007, le grand visionnaire Steve Ballmer (mode troll) se moquait et rigolait : 500 dollars ? Pas de clavier, pas attirant pour les professionnels, pas une très bonne machine. Mais nous, Microsoft, nous avons des terminaux peu chers et Windows Mobile…

On connaît la suite. À la décharge de Ballner, d'autres gros acteurs de la téléphonie mobile ont raillé Apple et n'y voyaient aucune menace, citons notamment BlackBerry et Nokia. Dix ans plus tard, ils n'existent plus ou quasiment.

Microsoft a mis du temps à remettre à plat son système mobile. Windows Phone 7 avait ouvert une nouvelle voie intéressante sur les services et l'interface. Mais c'est surtout Windows Phone 8 qui va être le véritable renouveau. Et il faut avouer qu'il était attrayant, mais le système ne pouvait s'installer sur les terminaux sous Windows Phone 7.5 et laissait donc de côté toute une partie du parc installé. En 2013, Microsoft met la main sur la division mobile de Nokia pour 5 milliards et espère ainsi concurrencer Apple et Google.

La gamme Lumia, dont les modèles haut de gamme, sont bien finis et proposent de belles fonctionnalités. Même si ces modèles réussissent localement ici et là, notamment en France, à dépasser les 10 % de marché, Windows Phone n'arrive pas à s'imposer et encore moins à être cette fameuse 3e plateforme. En 2014, nous avons même droit à un Lumia sous Android avec une surcouche d'interface à la Windows Phone, Nokia X !

Le constat est amer : Windows Phone / Mobile manque d'apps et les développeurs boudent. Les errements de la stratégie ont fait mal avec des sorties matérielles de plus en plus rares puis gelées, des licenciements massifs, la revente de Nokia, la disponibilité d'apps et de services sur Android et iOS…



Microsoft ne fut pas le seul à s'y casser les dents. Nous pouvons citer les projets Meegoo et Tizen, soutenus et développés par Intel et Samsung. Ubuntu s'aventura sur les OS mobiles avec Ubuntu Touch, tout comme Mozilla avec Firefox OS. Deux tentatives qui se soldent par des échecs.

Linux, open source, logiciels libres

Linux apparaît en 1991. Linus Torvalds développe un noyau répondant aux spécifications POSIX. Rapidement, il ouvre le projet aux développeurs extérieurs pour l'aider. L'open source était né. Ce noyau ne s'appelle pas encore Linux, mais Freax.

Le projet connaît un boost sans précédent quand les grands éditeurs utilisent Linux ou portent des logiciels stratégiques comme la base Oracle en 1998. La même année, la distribution française Mandrake apparaît. Toujours cette année-là, la première version de KDE, interface graphique pour Linux, sort, quelques mois plus tard, ce sera au tour de Gnome.

Durant les années 2000, Linux s'impose peu à peu sur le serveur. Et l'open source fait sa place dans les entreprises, les utilisateurs et les développeurs. Aujourd'hui, utiliser des frameworks / outils open source est naturel et de bon sens. Il y a 15 ans, ce n'était pas le cas. Une véritable opposition entre logiciels propriétaires – open source & logiciels libres existent. Mais peu à peu, cette guerre tourne à l'avantage du monde ouvert, surtout quand les éditeurs commencent à investir dans les communautés et les fondations. IBM est un des premiers à annoncer un investissement de long terme et plusieurs outils sont reversés à la communauté, un des exemples est Eclipse.

Eclipse fut initié et développé par IBM. Sun le regardait avec méfiance. À l'automne 2003, IBM décide de libérer Eclipse. En janvier 2004, la fondation Eclipse est créée avec pour mission de gouverner Eclipse et de ne plus dépendre d'IBM. Très rapidement, des dizaines d'éditeurs rejoignent le mouvement. Aujourd'hui, la fondation Eclipse est l'une des plus importantes organisations open source au monde avec la fondation Linux (2007) ou Apache (1999).

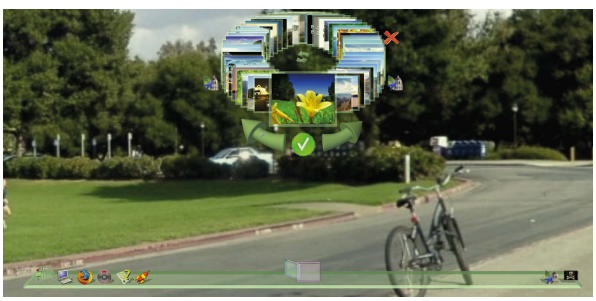
N° 98 juin 2007 : PHP 6, les nouvelles fonctions, ce qui change, migrer de PHP 5 à 6

La communauté PHP regardait la future version, la v6. Les retards s'accumulaient au fur et à mesure que les mois passaient. PHP 6 promettait beaucoup de changements et des ruptures radicales : + rapide, Unicode, meilleure sécurité. Voilà les trois mots d'ordre. Le support d'Unicode, système d'encodage des langues, était le focus le plus attendu. Mais ce support influait sur l'ensemble de PHP ce qui n'allait pas sans difficulté. L'autre grosse nouveauté était l'apparition des espaces de nom comme dans les autres langages « modernes ». Initialement, ils étaient prévus dans PHP 5. La sensibilité à la casse était planifiée pour PHP 6.1.

Finalement, le projet sera abandonné. PHP passera directement à la version 7 pour éviter toute confusion. Elle sortira en décembre 2015.

Looking Glass : une interface 3D Java qui n'intéresse pas grand monde

En 2003, Sun dévoile une nouvelle interface qui se veut révolutionnaire : Looking Glass. Il s'agit d'une interface 3D pour Linux, Solaris et Windows. Elle s'appuie sur Java 3D. L'idée est de changer les paradigmes de l'interface graphique que l'on avait sur les systèmes desktop : navigateur dans l'espace, affichage 3D. Exercice de style pour les uns, véritable intérêt pour les autres, le projet a suscité beaucoup de débats. La version 1.0 est sortie en 2006.

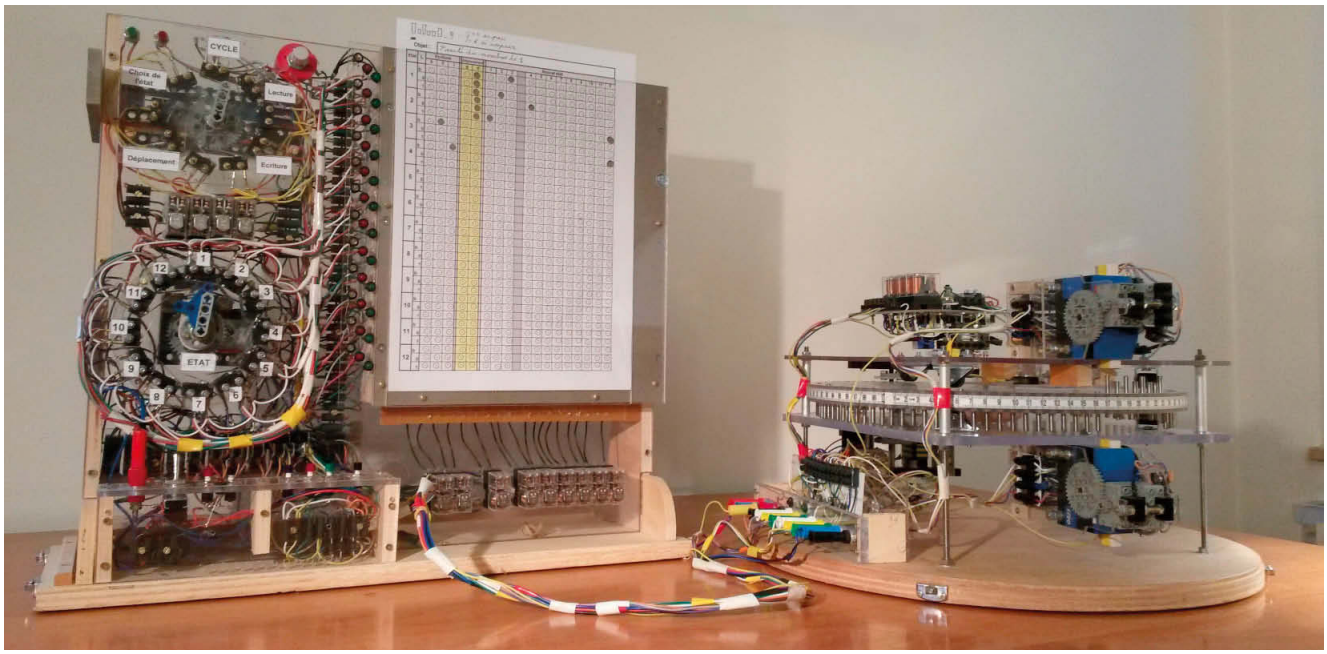


Quelques mises à jour furent proposées, mais le projet n'ira pas beaucoup plus loin.

La suite de notre rétrospective dans le n° 219.

Marc RAYNAUD
professeur de mathématique à la retraite (*)
Ayant construit très jeune de nombreux systèmes électriques, j'ai pu réaliser ce prototype en utilisant principalement les technologies des années trente, à savoir : circuits électriques, relais et moteurs électriques asservis par des systèmes à cames.

La machine de Turing



Alan Turing est né à Londres le 23 Juin 1912, à la fin de ses études au King's College de Cambridge, il va s'intéresser à la logique mathématique notamment sous l'influence de Max Newman.

Pour répondre à un problème posé par David Hilbert en 1928 : « est-il possible de trouver une méthode effectivement calculable pour décider si une proposition mathématique est démontrable ou non ? » Alan Turing publie en 1936 un article fondateur intitulé : « On Computable Numbers, with an Application to the Entscheidungsproblem» Dans cet article, il décrit un système que son maître de thèse à Princeton, Alonzo Church, appellera la machine de Turing; c'est l'objet de cet article. Pendant la guerre, il participe au décryptage des messages allemands destinés à la marine et codés avec la machine Enigma. Après la guerre il s'intéressera à la morphogenèse puis imaginera ce qu'on appelle le test de Turing : une personne pose des questions et doit déterminer si c'est un ordinateur ou un humain qui lui répond, Turing prévoit que 50 ans plus tard les ordinateurs pourront tromper un grand nombre de personnes… on n'en est pas loin !

Sa fin est tragique, condamné pour homosexualité, il sera soumis à un traitement hormonal castrateur, il finira par se suicider en Juin 1954 à 42 ans en mangeant une pomme enduite de cyanure. C'est la fameuse pomme croquée de Turing qui sera à l'origine du logo d'Apple. En 2013, la reine d'Angleterre le reconnaît comme héros de guerre et le gracie à titre posthume.

La machine de Turing

Elle est constituée d'un ruban infini, divisé en cases pouvant contenir des symboles.

Une tête de lecture/écriture peut :

- Lire le symbole écrit dans la case située en dessous ;
- Effacer le symbole ou en écrire un autre ;
- Se déplacer d'une case à gauche ou d'une case à droite ;
- Changer d'état.

Pour terminer, le système comprend une gestion des états, ce qui nous permettra de la programmer.

Selon le schéma 1, la machine est dans l'état 3 et lit un 1, on peut par exemple lui donner comme instruction :

- Ecris 0
- Déplace-toi d'une case à gauche
- Passe à l'état 5.

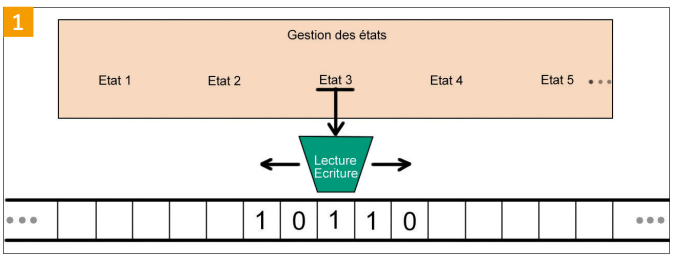
Voilà vous savez tout… La première impression est que l'on ne va pas pouvoir faire grand-chose avec ça… Grave erreur, le système est plus puissant que tous vos ordinateurs.

La programmation ne nécessitera aucun vocabulaire particulier, il suffira d'indiquer à la machine ce qu'elle doit faire en fonction de son état et du résultat de la lecture, il en résulte ce que l'on appelle une table des transitions dont voici un exemple :

Table des transitions

Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b		Droite	3
	0		Gauche	
	1	0	Droite	2
2	b	0	Gauche	1
	0		Droite	
	1			
3	b			12 (Final)
	0	1	Droite	
	1			

1



La colonne lecture contient tous les symboles que la machine peut lire, ici il s'agit simplement de b, 0 et 1. Vous trouvez sur fond jaune les éléments de la programmation : ce que la machine doit écrire, les déplacements demandés et éventuellement le nouvel état.

La réalisation du prototype

Le ruban :

Ne pouvant construire un ruban infini, j'ai replié le ruban sous la forme d'un disque muni de 100 cases, il est quand même illimité. 2

Technique : Le diamètre d'un disque d'un mètre de circonférence est égal à 1/ π … infinité de décimales. J'ai contourné le problème en réalisant un disque d'une circonférence très légèrement supérieure à un mètre et j'ai utilisé un mètre de couturière qui peut s'allonger un peu en l'étirant, il restait à faire correspondre la graduation 100 avec celle de 0.

Les symboles :

Dans vos ordinateurs, les nombres sont écrits sur des espaces toujours les mêmes, par exemple 32 bits, il n'est pas nécessaire d'avoir des séparateurs entre les nombres. Mais avec une tête de lecture qui ne se déplace que d'une case à gauche ou à droite, il me fallait un séparateur. J'ai donc opté pour le système binaire auquel j'ai rajouté un espace noté ici b (pour blanc).

L'alphabet de ce prototype est donc {b,0,1}

Les symboles sont représentés par des petits cylindres qui peuvent prendre 3 hauteurs :

b : le cylindre est complètement enfoncé

0 : le cylindre est à mi-hauteur

1 : Le cylindre est totalement relevé.

Technique : pour obtenir un frottement doux lors du déplacement des cylindres, j'ai opté pour des petits aimants qui exercent un force légère et perpendiculaire aux cylindres.

Écriture : 3

Deux petits moteurs Lego entraînent des crémaillères qui poussent les petits cylindres soit vers le bas soit vers le haut.

Technique : les moteurs sont asservis par des cames et des relais, le mouvement des crémaillères est soit de 7mm soit de 14mm.

Lecture :

Une pièce en plexiglas est poussée contre les cylindres, la position de blocage indique à la machine le symbole écrit. 4

Technique : 3 bits de mémoire, 3 relais pour enregistrer le résultat de la lecture.

Déplacement du disque 5

Un moteur Lego entraîne un axe sur lequel sont fixées 2 roues dentées en plexiglas qui engrènent directement sur les cylindres. Il est plus facile de faire tourner le ruban sous forme de disque que de déplacer les têtes de lecture/écriture.

Technique : comme les petit cylindres peuvent être soit relevés soit enfoncés vers le bas, il faut 2 roues dentées pour agir dans les deux cas sur le disque !

Horloge du cycle 6

Elle génère des impulsions qui sont dirigées vers chacun des composants de la machine, c'est l'équivalent du quartz de vos ordinateurs.

Le commutateur des états : 7

De l'état initial 1 à l'état final 12 qui provoque l'arrêt de la machine, il redirige le résultat de la lecture vers les lignes correspondantes du système de lecture des feuilles de programmation (tables des transitions)

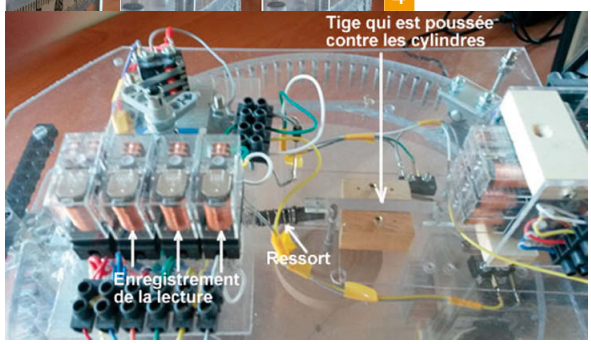
Écrire sur le disque 8

On peut soit utiliser le clavier de la machine soit déplacer à la main les petits cylindres sur le disque.

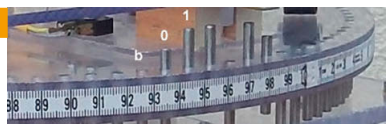
Schéma électrique général

Selon Alan Turing : « A tout instant ce que peut faire la machine est entièrement déterminé par le résultat de la lecture et l'état dans lequel elle se trouve.»

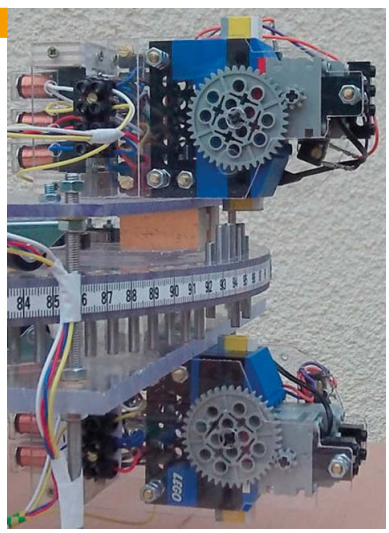
4



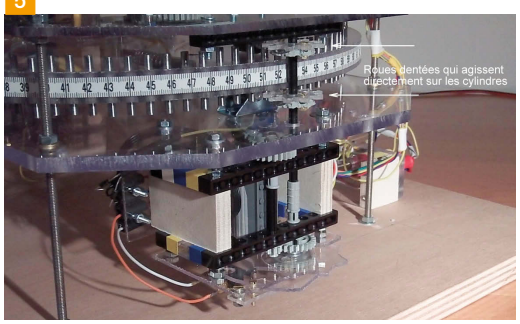
2

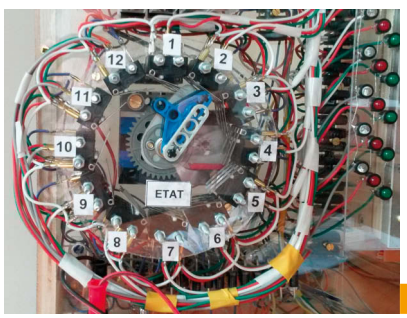


3

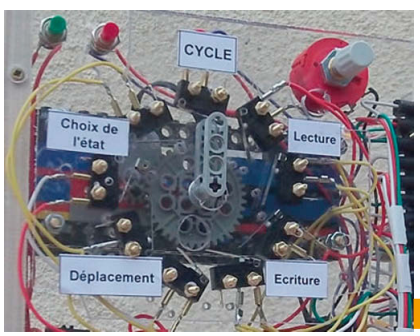


5

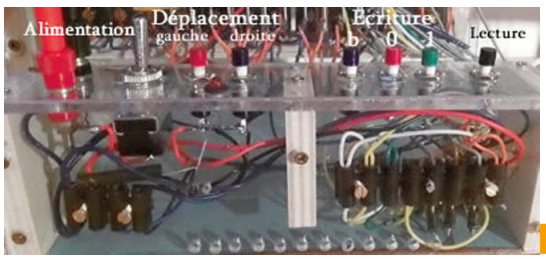




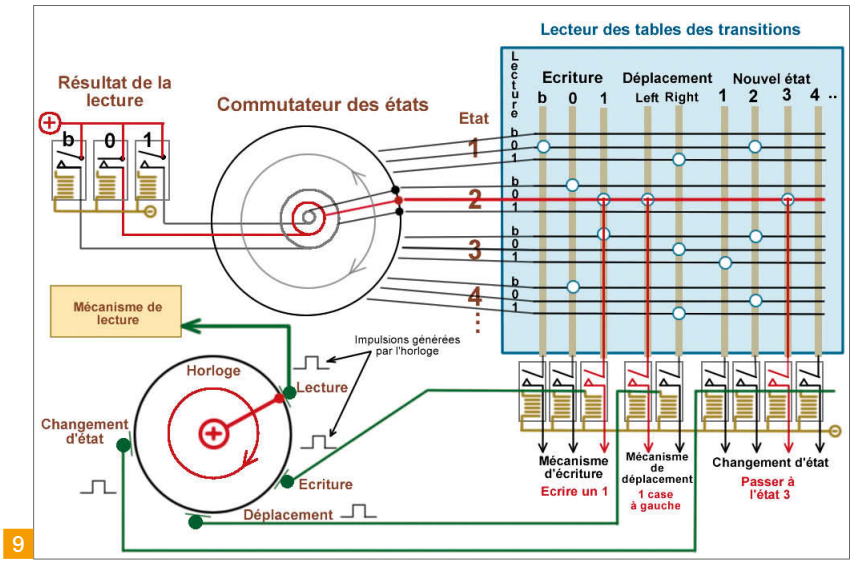
7



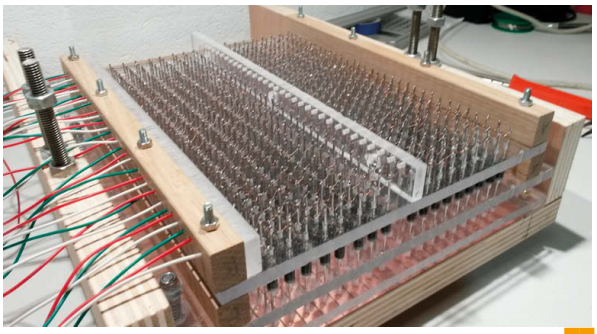
6



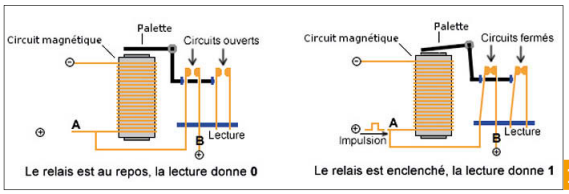
8



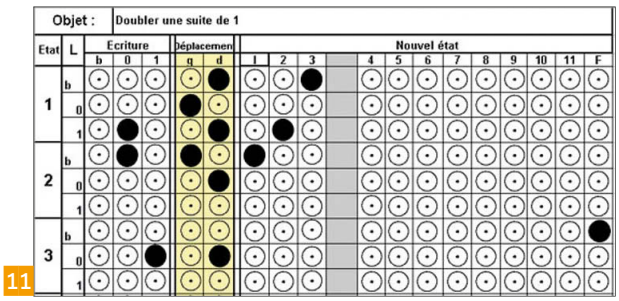
9



12



10



11

2 mémoires : lecture enregistrée dans un relais, état enregistré simplement par la position du commutateur. 9

Ici, le résultat de la lecture est 0 (suivre le fil rouge), la machine est dans l'état 2, cela donne une ligne précise de la table des transitions. Selon cette table, elle devra écrire 1, se déplacer d'une case à gauche et passer à l'état 3.

Technique :

Comment utiliser un relais pour représenter un bit de mémoire, le relais s'auto alimente sur le circuit +, c'est à dire qu'après avoir reçu une impulsion électrique sur la base A, il enclenche le premier contact qui relie cette base A au pôle + du circuit, il est donc alimenté par ce contact, on dit dans ce cas qu'il s'auto alimente. 10

Feuille de programmation : 11

C'est simplement une table des transitions dans laquelle on a percé des trous, elle sera lue par une presse comprenant 612 pointes, qui permettront, à travers les trous une connexion avec des bandes de cuivre pour conduire le courant et ainsi transmettre à la machine ce qu'elle doit faire. 12

Programmation de la machine

1er exemple

Trouver une séquence et remplacer les 0 par des 1, la tête de lecture/écriture étant à gauche de la séquence.

Pour cet algorithme, on utilise 2 états :

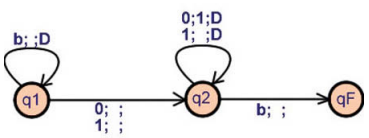
A l'état 1

- la tête devra se déplacer à droite tant qu'elle lira un blanc, dès qu'elle trouve un 0 ou un 1 elle passe à l'état 2.

A l'état 2

- si elle lit un 0, elle écrit 1 et se déplace à droite
- si elle lit un 1, elle n'écrit rien et se déplace à droite
- enfin, si elle lit un blanc, c'est qu'elle a fini de parcourir la séquence, elle passe à l'état final 12 et s'arrête.

Table des transitions				
Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b		Droite	
	0			2
	1			2
2	b			F
	0	1	Droite	
	1		Droite	



Conventions pour les graphes (*1)

Voilà le résultat :

2ème exemple

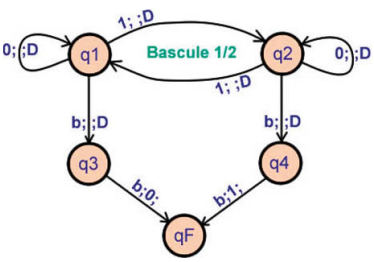
Parité du nombre de 1 dans une chaîne de caractères

Ici, c'est inutile de compter les 1, on va utiliser une bascule. La machine passera de l'état 1 à l'état 2 et réciproquement à chaque fois qu'elle lira un 1.

Elle termine à l'état 1 : il y a un nombre pair de 1

Elle termine à l'état 2, il y a un nombre impair de 1

Table des transitions				
Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b		Droite	3
	0		Droite	
	1		Droite	2
2	b		Droite	4
	0		Droite	
	1		Droite	1
3	b	0		12
	0			
	1			
4	b	1		12
	0			
	1			

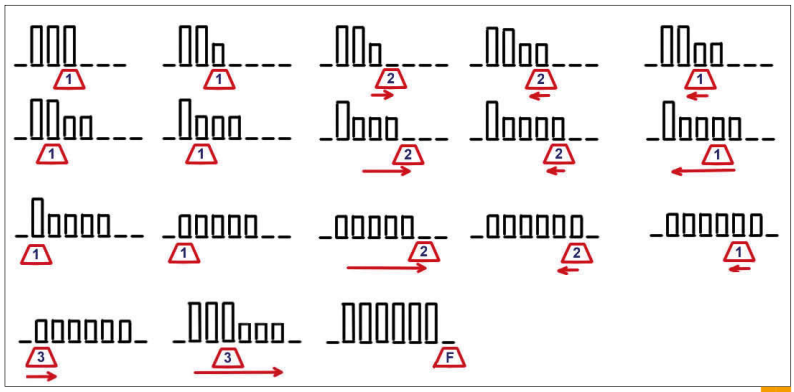


Le résultat :

3ème exemple

Doubler le nombre de 1

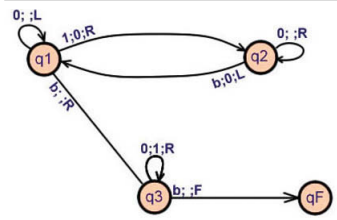
Pour que la machine se souvienne des 1



13

qu'elle a traité, elle va les remplacer par des 0.

Table des transitions				
Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b		Droite	3
	0		Gauche	
	1	0	Droite	2
2	b	0	Gauche	1
	0		Droite	
	1			
3	b			12 (Final)
	0	1	Droite	
	1			



Exécution du programme 13
Voir la vidéo sur le site de la machine, onglet vidéos.

4ème exemple : Calculs en binaire

Ajouter 1 à un nombre binaire

1+1 = 1 0

Table des transitions				
Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b	1		12 (F)
	0	1		12 (F)
	1	0	Gauche	

Le résultat : 1011 + 1 = 1100

Soustraire 1 à un nombre binaire
Il est supposé que le nombre binaire contient au moins un 1

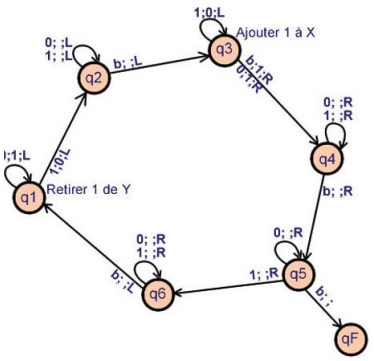
Table des transitions				
Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b			
	0	1	Gauche	
	1	0		12 (F)

Le résultat : 1100 – 1 = 1011

Addition de 2 binaires

On va ajouter Y à X, pour cela, on retire 1 à Y et on ajoute 1 à X jusqu'à ce que Y ne contienne plus de 1.

Table des transitions				
Etat	Lecture	Ecriture	Déplacement	Nouvel état
1	b			
	0	1	Gauche	
	1	0	Gauche	2
2	b		Gauche	3
	0		Gauche	
	1		Gauche	
3	b	1	Droite	4
	0	1	Droite	4
	1	0	Gauche	
4	b		Droite	5
	0		Droite	
	1		Droite	
5	b			12 (F)
	0		Droite	
	1		Droite	6
6	b		Gauche	1
	0		Droite	
	1		Droite	



Le résultat : 101 + 11 = 1000

Voir la vidéo sur le site de la machine, onglet vidéos.

*(*1) Conventions pour les graphes*

Les états sont nommés q1,q2, etc.
Les éléments sur les flèches sont dans l'ordre : Lecture ; écriture ; déplacement
Première flèche : à l'état 1, si la machine lit 1, elle écrit 0, se déplace à gauche (L) et reste à l'état 1
Deuxième flèche : à l'état 1 si la machine lit b ou 0, elle écrit 1, elle ne se déplace pas et elle passe à l'état final (12)

Utilisation de ce prototype :

Séances de recherche d'algorithmes par des étudiants ou des lycéens

Après une présentation générale de la machine, je propose une liste d'algorithmes à programmer. Les étudiants viennent eux-même tester leur solution sur la machine.

Conférences tout public

Rappel historique sur Turing, sur les problèmes mathématiques et les machines de l'époque avec mise en fonctionnement du prototype.
Adresse du site :
https://www.machinedeturing.org
Vous trouverez sur le site internet :

- Tous les détails sur sa fabrication ;
- De très nombreux programmes qui fonctionnent sur ce prototype ;
- De nombreuses vidéos qui vous montreront la machine en fonctionnement ;
- Un large éventail de mes interventions.

Remerciements

Merci à mon fils Olivier, enseignant chercheur en informatique à l'Université Blaise Pascal de Clermont Ferrand qui m'a demandé de construire ce prototype.
Tous mes remerciements aux membres de l'association Rennes en sciences qui me soutiennent dans mes activités.
Et un grand merci à François Tonic qui m'a proposé d'écrire cet article. •

(*) Marc Raynaud :
1981 : création d'une entreprise en énergies renouvelables
2001 : création d'une SSII : sites internet avec bases de données
Membre des associations : Rennes en sciences, Espace des sciences, Amelycor et Aseiste.
2017 : Présentation de la machine au centre Henri Lebesgue, voir la vidéo sur :
https://www.lebesgue.fr/fr/video/5min/raynaud

Laurent Ellerbach
Technical Evangelist Manager, Central and Eastern Europe,
Microsoft
laurelle@microsoft.com
http://github.com/ellerbach

Microsoft Quantum Development Kit :

Q#, un nouveau langage, des simulateurs, multiplateforme pour du quantique

© Rost-9D

Le développement quantique devient une réalité. Microsoft a créé un nouveau langage, Q#, des simulateurs, locaux et dans Azure, des libraires open source et de nombreux outils pour commencer à créer des applications utilisant les processeurs quantiques. Avant de rentrer dans les détails des outils, faisons un petit peu de théorie pour mieux comprendre les différents aspects des ordinateurs quantiques. Petit conseil : à lire à tête bien reposée !

Le chat de Schrödinger est-il mort ou vivant ?

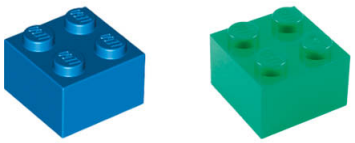
La fameuse expérience de pensée qu'Erwin Schrödinger a imaginée en 1935 permet d'expliquer les fondamentaux de la mécanique quantique. Le chat est enfermé dans une boîte avec un dispositif qui tue le chat quand survient une désintégration d'un atome d'un corps radioactif. Tant que la boîte est fermée, il est impossible de savoir si le chat est mort ou vivant.

L'objectif est de faire comprendre un des éléments fondamentaux de la mécanique quantique : la superposition quantique. Pour faire simple, une particule, comme un électron, peut se trouver à plusieurs endroits en même temps. La mécanique quantique est née des mathématiques. Et la position d'un électron s'exprime avec des probabilités. La probabilité qu'un électron se trouve à plusieurs endroits en même temps est justement cette superposition quantique. Vous me direz : quel est le rapport avec le chat dans tout ça ? Eh bien, la désintégration est totalement aléatoire et peut intervenir n'importe quand. Donc, il n'est pas possible de prévoir quand le chat sera tué par le mécanisme de la boîte. Et donc le seul et unique moyen de savoir si le chat est toujours vivant est d'ouvrir la boîte. Une fois la boite ouverte, on saura donc si le chat est mort ou s'il est vivant.

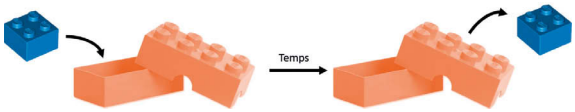
En informatique quantique, quand on veut connaître l'état du chat, on fait ce que l'on appelle une mesure. A ce moment-là, nous sommes certains de l'état du chat, il n'y a plus de mystère et le chat restera dans cet état.

De bit à qubit

Dans un ordinateur classique, on trouve des bits, ils ont un état défini, soit 0, soit 1. Mais ce choix est arbitraire. On pourrait très bien utiliser des briques de Lego de couleur, une bleue et une verte.

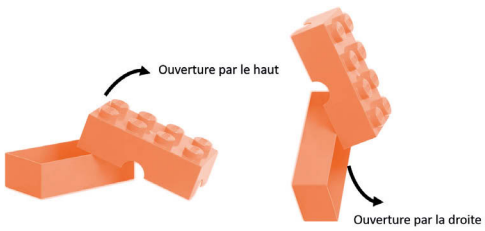


Imaginons maintenant que l'on place une des briques, disons la bleue dans une boîte Lego, que l'on referme la boîte et qu'on l'ouvre au bout d'un certain temps.



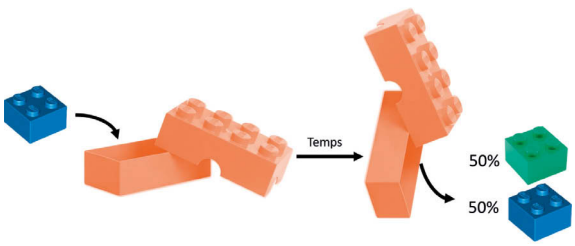
Avec un ordinateur classique, on est sûr que la couleur de la brique sera la même. Comme on est sûr également de la couleur de la brique, il est très facile de recopier la brique. Impossible de se tromper avec un ordinateur classique.

Dans un ordinateur quantique, les choses sont un peu différentes. D'abord, notre boîte Lego va être un peu spéciale, elle va avoir 2 ouvertures, une ouverture par le haut et une ouverture par la droite.



Nos briques Lego sont maintenant des qubriques (nos briques Lego quantiques) dans le monde quantique. Si l'on place une qubrique par la porte du haut et que l'on attend, puis que l'on ouvre la porte du haut, alors on trouvera une brique de la même couleur, comme avec nos briques classiques.

Là où cela devient intéressant (et frustrant pour l'esprit) c'est quand on met nos qubriques dans la boîte Lego par le haut et que l'on regarde le résultat par la porte de droite.



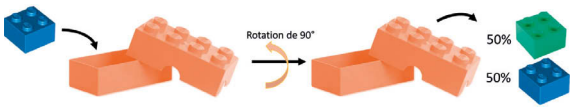
Dans ce cas, on a autant de chance de trouver la qubrique verte ou bleue. La probabilité est donc ½. C'est exactement ce qui se passe avec le chat, c'est le principe de la superposition quantique.

Mais une fois la boite ouverte, on a une qubrique soit verte soit bleue. Et on est certain qu'elle est soit verte soit bleue. Il n'y a plus de superposition. La lecture, donc la mesure, détruit la superposition quantique.

Allez, on continue dans les étrangetés. Disons que je mets une qubrique dans une des ouvertures, soit celle du haut, soit celle de droite sans vous le dire. Et que je vous demande de recopier la couleur de la qubrique. Si je mets disons la bleue en haut et que vous ouvrez la boîte par le haut, bingo, vous aurez une qubrique bleue et vous pouvez la recopier. Mais si vous ouvrez la boîte par la droite, vous aurez autant de chance d'avoir une qubrique verte que bleue. Du coup, impossible pour vous d'être sûr, vous avez 1 chance sur 4 de faire une erreur.

Vu qu'avec un ordinateur classique, on passe notre temps à recopier des informations, cela va être compliqué de faire des calculs juste avec un ordinateur quantique vu qu'en recopiant une information, on a une chance sur 4 de se tromper ! Mais gardons cela de côté pour plus tard.

Dans un ordinateur quantique, on ne va pas ouvrir la boîte par une autre porte mais on va effectuer une rotation de 90° de la boîte et on ouvrira la boîte par le haut.



Cette rotation est un des composants importants de la programmation quantique et nous la retrouverons plus tard lorsque nous écrirons du code quantique. Cette rotation est nécessaire car dans un ordinateur quantique, il n'est possible de n'ouvrir que la porte du haut mais il est aussi possible d'effectuer des rotations.

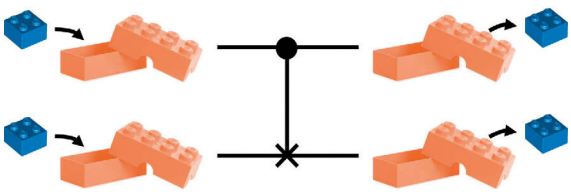
Si nous repassons dans le monde de l'ordinateur classique et de l'ordinateur quantique, avec ce que nous avons vu précédemment, nous avons donc besoin d'une notion plus compliquée que la simple brique de couleur bleue ou verte, la qubrique avec sa probabilité d'être bleue ou verte. Du coup, nous parlerons de bit avec des ordinateurs classiques et de qubit avec des ordinateurs quantiques.

Intrication quantique es-tu là ?

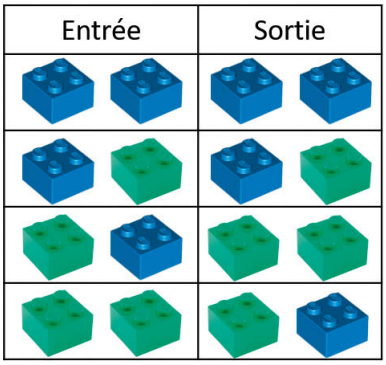
Continuons notre chemin dans la théorie pour arriver à une autre notion qui est l'intrication quantique. Et pour cela, nous allons nous intéresser aux portes logiques. Dans un ordinateur classique, ce sont les portes NON, ET, OU par exemple. Avec l'ordinateur quantique, nous aurons aussi des portes mais elles sont un peu particulières.

Première particularité, elles sont toutes réversibles. Le monde quantique est un monde d'abord décrit par les mathématiques et les probabilités, du coup tous les opérateurs quantiques sont des opérateurs linéaires. Et comme ils sont linéaires, cela implique qu'un nombre exponentiel d'opérations peuvent être effectués en parallèle. C'est ce qui excite tellement les esprits et crée tellement de mythes autour de l'informatique quantique. Mais calmons nos ardeurs, continuons un peu la théorie.

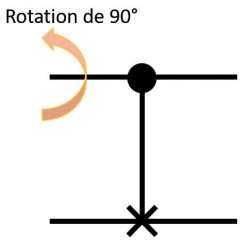
Une des portes les plus importantes dans l'informatique quantique est la porte CNOT (Controlled-Not Gate). Avec nos qubriques, elle se représente comme cela et prend 2 qubriques en entrée.



Et les états d'entrée et sortie sont les suivants :



En regardant de près les résultats, on s'aperçoit que l'intérêt de cette porte est d'inverser l'état de la seconde qubrique si la première qubrique est verte. Bon, juste là, on se dit qu'il n'y a rien de bien intéressant. Passons donc à l'étape suivante. Ajoutons donc à la première qubrique une rotation de 90° :



Que se passe-t-il maintenant si on prend nos qubriques en entrée ? Quelle que soit la couleur de la qubrique sur la première entrée, une fois la rotation effectuée, j'ai ½ chance d'avoir soit une qubrique bleue soit une verte. Partons du principe que j'ai une qubrique bleue en seconde entrée et d'après le tableau de la porte CNOT:

- Si la qubrique de la première entrée est bleue, alors j'aurai 2 qubriques bleues en sortie
- Si la qubrique de la première entrée est verte, alors j'aurais 2 qubriques vertes en sortie

Et donc si vous avez bien suivi, avec cette suite de portes, nous arrivons donc à avoir simultanément 2 qubriques bleues ou 2 qubriques vertes. Et si nous décidons maintenant de mesurer le résultat, la couleur d'une seule des 2 qubriques donnera la couleur de l'autre qubrique car ce sont forcément les mêmes.

Et c'est cette propriété que l'on appelle l'enchevêtrement quantique ou l'intrication quantique.

Bon, c'est bien étrange tout ça mais ça sert à quoi ? Et bien cela sert à téléporter des informations. Oui, vous avez bien lu téléporter. Une fois les 2 qubriques intriquées telles qu'avec les 2 portes, une qubrique peut rester à un endroit et l'autre voyager à l'autre bout de l'univers. A la lecture d'une des deux qubriques, il est possible de déterminer quelle est la couleur de l'autre qubrique de façon certaine. Bon, là, je vois que votre cerveau commence à tiquer. Vous vous souvenez de la théorie de la relativité restreinte, qu'il n'est pas possible de faire voyager une information plus rapidement que la lumière. En réalité, il n'y a pas d'information échangée. L'état d'origine détermine l'intrication, il n'y a pas d'information qui circule. Alain Aspect, a réalisé des expériences dans les année 1980 dans son laboratoire à Orsay qui prouvent cela et a obtenu le prix Nobel.

Et comme c'est amusant la téléportation, c'est donc l'exemple de code que nous utiliserons plus tard dans l'article.

De l'ordinateur classique au quantique

Sur un ordinateur classique à 3 bits, il y a 2x2x2 = 8 combinaisons possibles car chaque bit a uniquement 2 combinaisons possibles, 0 ou 1, brique bleue ou brique verte. Sur un ordinateur classique de 7 bits, il y a 2x2x2x2x2x2x2 = 128 combinaisons possibles parce qu'un bit est toujours un bit et qu'il n'a que 2 combinaisons possibles. Jusque-là, tout va bien.

Passons maintenant aux qubits. Un qubit a plusieurs combinaisons possibles car il peut être de couleur bleue, verte et comme on l'a vu avant, les qubits peuvent être en superposition d'état et donc si l'on veut effectuer le même calcul, on se retrouve avec 2^qubit possibilités de combinaisons par qubit. Donc à chaque fois que l'on ajoute un qubit, il faut doubler la taille nécessaire pour pouvoir adresser tous les états.

Ainsi, le simulateur livré avec le SDK permet, sur un PC classique de type Surface Book, de faire tourner 30 qubits utilisant 16 Go de mémoire. Si j'ajoute 1 qubit, alors, la mémoire nécessaire devient 32 Go et si j'enlève 1 qubit, alors je n'ai besoin que de 8Go de mémoire. Et si je décide d'utiliser 40 qubits, alors il va me falloir 16 To de mémoire ! Et avec 70 qubits, il faudrait 16 Zo. 1 Zo = 1 Zeta octet = 10^21 octets donc beaucoup beaucoup de mémoire.

Il faut à cela ajouter le temps pour simuler une porte, le temps est de l'ordre de la seconde sur un ordinateur classique pour 30 qubits mais de l'ordre de la seconde sur un supercalculateur pour 40 qubits.

C'est pourquoi si vous avez des simulations importantes à faire, vous pouvez utiliser Azure où un simulateur pouvant faire tourner des quantités plus importantes de qubits est disponible, et ainsi aller au-delà de la limite des 30 qubits du simulateur classique.

Il y a quand même des limites, pour simuler un ordinateur quantique de 260 qubits, il faudrait utiliser chaque atome de l'univers visible comme élément de stockage pour simuler la mémoire nécessaire et la simulation d'une porte prendrait l'âge de l'univers. Besoin d'une aspiration ou tout va toujours bien ?

Comme cela va toujours bien, continuons. La limitation du simulateur local permet donc de faire tourner uniquement des programmes appelant des fonctions quantiques avec des quantités limitées de qubits. Mais que je vous rassure, pour des exemples simples, c'est largement suffisant ! Et en attendant les puces spécifiques, cela permet de mettre au point des programmes relativement simples. Et gardez en mémoire que ce ne sont que des simulateurs et par définition ne permettent pas de refléter exactement un ordinateur quantique.

Et c'est en regardant un peu plus près les fonctions disponibles sur les ordinateurs quantiques que l'on se rend bien compte que nos ordinateurs classiques ont encore une longue vie devant eux ! Il faut voir l'ordinateur quantique comme un coprocesseur, comme le processeur additionnel d'une carte graphique permettant d'effectuer des tâches spécifiques pour lequel il est particulièrement performant et non l'ensemble des tâches que le processeur actuel et son environnement permet d'effectuer.

L'intérêt est surtout quand il s'agit de résoudre des problèmes où le calcul sur un ordinateur est exponentiel alors qu'il l'est en mode linéaire ou quasi linéaire sur un ordinateur quantique. Par exemple une transformée de Fourrier rapide (FFT, https://en.wikipedia.org/wiki/Fast_Fourier_transform) utilise O(n * 2 ^ n) opérations où n est le nombre de bits. Une Transformée de Fourrier quantique (QFT, https://en.wikipedia.org/wiki/Quantum_Fourier_transform) elle

n'utilise que O(n ^ 2) opérations. Et comme vos souvenirs en maths sont encore bons, le nombre d'opérations nécessaires pour une QFT est donc exponentiellement moins important qu'avec des opérations classiques sur une FFT. Ce sont des mathématiques assez compliquées mais elles sont fondamentales dès que l'on va vouloir écrire ses propres algorithmes quantiques. Et quand on sait que les FTT sont utilisées intensivement dans les algorithmes de deep learning et autres, on se dit qu'il y a une grosse marge de progrès pour tout ce qui touche au machine learning avec les ordinateurs quantiques.

Autre exemple, la recherche de matériaux supra conducteurs à température ambiante en résolvant l'équation de Schrödinger qui est impossible à résoudre avec les moyens de calculs classiques. Il faudrait quand même disposer d'un ordinateur quantique d'environ 200 qubits. Et en l'état actuel des prototypes fournissant seulement quelques qubits, nous en sommes encore loin. Mais la direction est donnée et permettra de résoudre des équations impensables aujourd'hui, d'effectuer des simulations précises, de prévoir le climat plus précisément, de trouver des nouvelles molécules pour fixer l'azote et bien d'autres choses encore que nous n'imaginons pas encore.

Ce que contient le Microsoft Quantum Development Kit

Le Quantum Development Kit qui est actuellement en preview et contient aussi bien le langage Q# que les simulateurs. Le tableau ci-dessous récapitule les différents composants :

Composant	Fonction
Langage Q# et compilateur	Q# est un langage de programmation dit « domain-specific » ce qui implique qu'il n'est utilisable que pour le développement d'algorithmes quantiques. Il est utilisable pour écrire des sous-routines quantiques utilisables et pilotables depuis un langage traditionnel sur un ordinateur classique. Encore une fois, il faut voir l'ordinateur quantique comme un coprocesseur physique ou distant d'un ordinateur classique.
Librairies Q# standard	Les libraires contiennent les opérations et les fonctions pour supporter à la fois l'exécution des algorithmes quantiques et l'interface avec les langages classiques.
Simulateur machine quantique local	Un simulateur de vecteur optimisé pour des simulations les plus justes possibles et rapides. Gardez en tête que ce n'est qu'un simulateur et pas une machine quantique physique. Il y a donc des compromis qui ont été effectués.
Un simulateur de trace quantique	Le simulateur de trace ne simule pas en tant que tel l'environnement quantique mais il est utilisé pour estimer les ressources nécessaires à chaque étape ainsi que faciliter le débogage avec l'environnement classique. C'est un élément très important pour déterminer le nombre de qubits nécessaires pour faire tourner le programme.
Visual Studio extension	Une extension Visual Studio contenant les modèles (templates) pour le support des fichiers Q# ainsi que les projets et la syntaxe colorifique.
Visual Studio Code extension	Une extension Visual Studio Code qui contient la syntaxe colorifique et des code snippets pour Q#.

Pour pouvoir l'utiliser, vous devez installer soit Visual Studio, quelle que soit la version, y compris community, soit VS Code sur Windows, Linux ou Mac. Assurez-vous également d'avoir la toute dernière version de .NET Core 2 installée. Dans les versions antérieures, vous pouvez avoir des erreurs de compilation.

Attention, le simulateur n'est disponible que sur les versions 64 bits des PC et attention au nombre de qubits que vous utilisez, cf. le point précédent, vous pouvez très rapidement saturer votre machine.

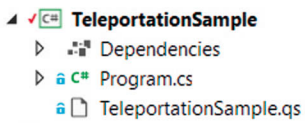
Lors de l'installation du Kit (http://www.microsoft.com/quantumdevkit), téléchargez également les projets sur GitHub (https://github.com/microsoft/quantum). J'utilise un de ces exemples dans le reste de cet article.

Du code, du qubit !

Je commence à comprendre que vous trépignez maintenant et que vous voulez voir du code. Alors c'est parti !

Comme indiqué, la partie quantique, ici écrite en Q# est à voir comme un composant additionnel et donc utilisable (même qui doit être utilisé) depuis une application classique. J'utiliserai C# mais il est possible aussi de le faire en Python, F# ou d'autres langages.

Comme indiqué, nous allons réaliser une fonction de téléportation quantique. J'utilise l'exemple du GitHub Quantum. Vous y trouverez beaucoup d'autres exemples. L'exemple comprends 2 fichiers, un fichier *Program.cs* contenant le code C# permettant notamment l'affichage des résultats, l'appel aux fonctions. Et *TeleportationSample.qs* contenant le code quantique proprement dit écrit en Q#



Commençons par le code C# :

```
using Microsoft.Quantum.Simulation.Simulators;
using System.Linq;

namespace Microsoft.Quantum.Examples.Teleportation {
  class Program
  {
    static void Main(string[] args)
    {
      using (var sim = new QuantumSimulator())
      {
        var rand = new System.Random();
        foreach (var idxRun in Enumerable.Range(0, 8))
        {
          var sent = rand.Next(2) == 0;
          var received = TeleportClassicalMessage.Run(sim, sent).Result;
          System.Console.WriteLine($"Round {idxRun}:\tSent {sent},\tgot {received}.");
        System.Console.WriteLine(sent == received ? "Teleportation successful!!\n" : "\n");
        }
      }
      System.Console.WriteLine("\n\nPress Enter to continue...\n\n");
      System.Console.ReadLine();
    }
  }
}
```

Il est simple, commence par référencer le simulateur, crée un namespace *Microsoft.Quantum.Examples.Teleportation* qui est le

même que dans le fichier qs que l'on verra juste après. Un simulateur est créé avec la variable *sim*. Ensuite 8 itérations sont effectuées. Pendant chaque itération, un message *sent* est envoyé. *sent* est un booléen créé aléatoirement.
La fonction *TeleportClassicalMessage* est appelée et le résultat est comparé à ce qui a été envoyé. Si la téléportation a réussi, alors on affiche le message de succès.
Voyons maintenant le code Q#. Premier constat quand vous ouvrez les fichiers Q# des exemples et des librairies, ils sont très bien documentés et contiennent des informations détaillées sur les algorithmes utilisés, des liens sur les articles scientifiques permettent de mieux comprendre le fonctionnement pour ceux qui veulent se mettre à écrire de vrais algorithmes quantiques. Regardons notre fonction *TeleportClassicalMessage* appelée dans le code C# :

```
operation TeleportClassicalMessage(message : Bool) : Bool {
  body {
    mutable measurement = false;
    using (register = Qubit[2]) {
      // Ask for some qubits that we can use to teleport.
      let msg = register[0];
      let there = register[1];

      // Encode the message we want to send.
      if (message) { X(msg); }

      // Use the operation we defined above.
      Teleport(msg, there);

      // Check what message was sent.
      if (M(there) == One) { set measurement = true; }

      // Reset all of the qubits that we used before releasing
      // them.
      ResetAll(register);
    }
    return measurement;
  }
}
```

Voici donc enfin du code en Q#. Premier constat, il semble facile à lire, ressemble à tous les langages modernes, une sorte de mix entre du C# et du VB. La déclaration de variables se fait à l'aide de l'instruction *let*. On a des *operations* et non des *functions*. On retrouve un *using*, des *if*, un *return*. Bref, ce n'est pas dans le langage en tant que tel qu'il faut chercher la complication. La complication se trouve dans les fonctions quantiques et leur utilisation.

Avant de rentrer dans le détail du code, on aperçoit une autre fonction qui s'appelle *Teleport*. Voyons le code également :

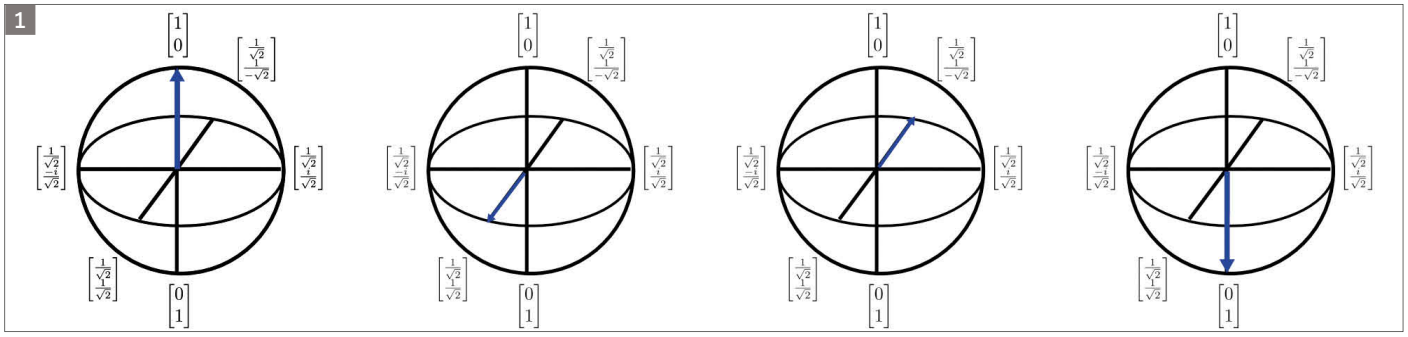
```
operation Teleport(msg : Qubit, there : Qubit) : () {
  body {

    using (register = Qubit[1]) {
      // Ask for an auxillary qubit that we can use to prepare
      // for teleportation.
      let here = register[0];

      // Create some entanglement that we can use to send our message.
      H(here);
      CNOT(here, there);

      // Move our message into the entangled pair.
      CNOT(msg, here);
      H(msg);

      // Measure out the entanglement.
      if (M(msg) == One)  { Z(there); }
      if (M(here) == One) { X(there); }

      // Reset our "here" qubit before releasing it.
      Reset(here);
    }

  }
}
```

A la lecture de ce code, on trouve des fonctions coloriées en bleues, donc faisant partie du langage telles que *H* et *CNOT* mais aussi *M*. Et comme vous vous souvenez de ce que j'ai écrit précédemment vous reconnaissez *CNOT*. La fonction *H* pour transformée de Hadamard, permet d'effectuer une rotation, telle qu'expliqué dans la partie théorique de l'article. Et *M* permet la mesure.
Pour pouvoir maintenant expliquer l'algorithme, il me faut refaire un peu de théorie et notamment sur la représentation des qubits.

Qubits, matrices et sphères de Bloch

Lorsque j'ai présenté les concepts j'ai parlé de tourner de 90° la boîte avec les qubriques. En réalité, il s'agit d'une transformation mathématique sur un vecteur de norme 1. Les qubits se représentent effectivement avec des vecteurs $\begin{bmatrix}\alpha\\ \beta\end{bmatrix}$ où α et β sont des nombres complexes qui satisfont à la formule suivante : $|\alpha|^2+|\beta|^2=1$. Cela implique que les vecteurs tels que $\begin{bmatrix}1\\ 0\end{bmatrix}$, $\begin{bmatrix}0\\ 1\end{bmatrix}$ ou encore $\begin{bmatrix}1/\sqrt{2}\\ 1/\sqrt{2}\end{bmatrix}$ mais aussi $\begin{bmatrix}i/\sqrt{2}\\ i/\sqrt{2}\end{bmatrix}$ sont aussi des vecteurs valides (i =

1



nombre imaginaire). Et comme on a besoin de correspondance pour les bits normaux, on utilise la convention suivante :
0 se représente $\begin{bmatrix}1\\0\end{bmatrix}$ et 1 comme $\begin{bmatrix}0\\1\end{bmatrix}$. C'est une convention et toutes les autres représentations de qubits n'ont aucun équivalent dans le monde classique binaire.
Maintenant quand il s'agit de faire une mesure, on va en fait mesurer la probabilité du vecteur [α, β] d'être 0 ou 1. La probabilité 0 est donc $|\alpha|^2$ et celle de 1 est de $|\beta|^2$. On s'aperçoit aussi que les signes d'α ou β n'ont aucune importance dans la détermination des états classiques 0 ou 1.
Et on s'aperçoit aussi que la mesure de $\begin{bmatrix}1\\0\end{bmatrix}$ (donc 0 en qubit) et $\begin{bmatrix}0\\1\end{bmatrix}$ (donc 1 en qubit) n'endommage pas la mesure. Et que dans ce cas et ce cas seulement, il est possible de copier sans aucun problème les données qu'ils contiennent. Mais dans le cas d'un vecteur $\begin{bmatrix}1/\sqrt{2}\\1/\sqrt{2}\end{bmatrix}$, la mesure détruit l'état car on obtiendra 50% de chance d'avoir un 0 ou un 1 alors que cet état était autre chose avant la mesure. Et la mesure va forcer un de ces états. C'est cela qui implique qu'il ne faut pas mesurer des qubits quand on n'en n'a pas besoin et c'est cela qui interdit la copie. Il y a un théorème complet disponible pour les plus courageux (https://en.wikipedia.org/wiki/No-cloning_theorem).
Le qubit 0 est représenté par |0> et le qubit 1 par |1>. Et plus généralement une série de qubits se représente |ab> où a et b sont 2 qubits. Si nos qubits sont des vecteurs normés alors, il est possible de les représenter graphiquement sur une sphère. Elle est dite sphère de Bloch. Le vecteur étant de 2 dimensions mais représentant des nombres complexes, on a donc bien 3 axes possibles, l'axe des deux dimensions des réels et l'axe complexe. 1
Et du coup, les rotations que l'on effectue sur les qubits prennent du sens dans une représentation telle que celle-là. Le langage Q# va nous faciliter la tâche car il comprend directement les rotations et autres opérations sur ces représentations vectorielles de nos qubits.
Je ne m'attarde pas sur tous les aspects de multiplication de matrices, de transformation mais toutes les portes sont des transformations de matrices. Je ne parlerai que de quelques-unes dans l'explication du code ci-après.

Retour au code

Retour à notre code de téléportation et l'opération *Teleport*. Il est à noter que *there* doit impérativement être dans l'état |0> pour que la téléportation fonctionne. Cf l'exemple avec les qubriques.
Dans la première partie du code, on crée un qubit *here*. La partie importante c'est l'opération *H(here)*. H pour transformée d'Hadamard. Elle a pour but de tourner notre qubit pour le mettre dans l'état de superposition. Donc là où l'on a autant de chance de lire un 0 qu'un 1.
En d'autres termes, on effectue une multiplication de matrice où $H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$.
Le résultat est de transformer |0> en $\frac{|0>+|1>}{\sqrt{2}}$ et |1> en $\frac{|0>-|1>}{\sqrt{2}}$.
Cette transformation ne s'effectue que sur un seul qubit. Nous voilà donc avec notre rotation.
La seconde étape est donc de lier *here* et *there* ainsi que notre message *msg* et *here* à travers la double opération *CNOT*. J'ai déjà décrit plus haut la port CNOT. Pour ceux qui aiment les transformées de matrice, $CNOT = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$. Celle-ci s'effectue sur le contrôle de l'axe X, il en existe aussi pour l'axe Y et Z.
Puis d'effectuer une nouvelle rotation mais cette fois-ci sur *msg*.
Il reste les opérations de mesure de *msg* et de transformée sur l'axe Z *there* si le message est |1> ainsi que d'effectuer une transformée sur l'axe X de *there* si *here* est |1>.
Dans l'opération principale *TeleportClassicalMessage*, il reste à lire le résultat de *there* dans ce qubit qui contient les informations originales de *msg* puis de retourner le résultat avec *return*.
Cette série de transformées à travers ces portes standard, et les mesures permettent d'intriquer les qubits. L'utilisation de la superposition permet d'opérer ~~la magie nécessaire~~ les transformées nécessaires à la téléportation. Voyons maintenant le résultat quand on lance le programme : 2
Et donc sans surprise, la téléportation a fonctionné à chaque fois ; la lecture du qubit de destination donne bien le même résultat que le qubit d'origine.
Pour ceux qui sont intéressés, il y a beaucoup de littérature sur le sujet de la téléportation, je ne rentre pas dans les détails, cela serait bien plus long que cet article. A noter que dans les commentaires du fichier Q#, vous avez davantage de pointeurs sur d'autres téléportations. Et vous trouverez une description des portes quantiques principales sur https://en.wikipedia.org/wiki/Quantum_gate ainsi que leur implémentation et des informations complémentaires sur le langage Q# sur https://docs.microsoft.com/fr-fr/quantum/quantum-concepts-1-intro.

2

```
C:\Program Files\dotnet\dotnet.exe
Round 0:        Sent False,     got False.
Teleportation successful!!

Round 1:        Sent False,     got False.
Teleportation successful!!

Round 2:        Sent True,      got True.
Teleportation successful!!

Round 3:        Sent True,      got True.
Teleportation successful!!

Round 4:        Sent True,      got True.
Teleportation successful!!

Round 5:        Sent False,     got False.
Teleportation successful!!

Round 6:        Sent False,     got False.
Teleportation successful!!

Round 7:        Sent False,     got False.
Teleportation successful!!


Press Enter to continue...
```

Conclusion

Vous l'aurez compris, la programmation d'un ordinateur quantique est pour l'instant relativement complexe. Elle nécessite de très solides bases en algorithmie quantique. Son utilisation en est cependant très simplifiée par le Microsoft Quantum Development Kit. Même pour un non spécialiste, il est assez facile de lire le Q# (même sans forcément comprendre dans le détail tous les algorithmes) et surtout très facile à utiliser depuis un langage évolué tel que F#, C# ou Python.
Ce kit s'adresse donc principalement à ceux qui veulent utiliser des capacités de calcul quantiques et être prêts quand les ordinateurs arriveront. Soit en développant leurs propres algorithmes, soit en utilisant des algorithmes existants. Et vous pouvez suivre une série de vidéos vraiment bien faites sur le sujet sur http://aka.ms/QuantumPlaylist. Je vous recommande également de regarder tous les exemples fournis, ils sont détaillés et complets, ils couvrent de larges domaines d'utilisation.
On me demande souvent quand arriveront ces ordinateurs et surtout quand ils seront disponibles pour tout développeur. Selon les personnes à qui la question est posée, la réponse varie de 6 mois à 5 ans. Et c'est la réponse que j'ai depuis les 2 dernières années. Aussi, je serais assez conservateur et je dirais que les premiers utilisables avec un nombre raisonnable de qubits autour de 40 à 50 seront pour dans 3 à 5 ans, utilisables par certains développeurs. Et je pense qu'ils le seront avant tout dans le cloud, en mode serverless, avant de pouvoir se généraliser de type coprocesseur dans les 10 prochaines années. Quant aux ordinateurs de 100 à 200 qubits, ceux qui permettent vraiment de réaliser des calculs intéressants ne le seront pas avant 10 à 20 ans. Les informaticiens ont toujours été très (trop) optimistes quant à l'arrivée de nouvelles technologies de rupture. •

Une réalité aux multiples facettes

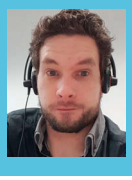
En Science Fiction, les réalités alternatives sont communes : Tron, Star Trek NG, Dark City / Matrix, Caprica, Battlestar Galactica, etc. Dans notre vie quotidienne, c'est relativement nouveau, même si ce n'est pas un monde totalement inconnu.

Par une étrange coïncidence, c'est en 2012 que Google termine le développement des lunettes augmentées, les Google Glass, et que Kickstarter accueille le projet Oculus. Et depuis, les casques, lunettes se sont multipliés ainsi que les accessoires.

Il était tout naturel de consacrer un grand dossier dans ce numéro un peu spécial de Programmez!. Dans cette première partie, nous avons fait un point sur la réalité augmentée / virtuelle, sur comment ces technologies vont révolutionner le web. Vous verrez aussi une mise en oeuvre de Vuforia, un dérivé d'OpenCV, et nous ferons un focus technique sur la technologie Windows Mixed Reality de Microsoft. Cette plateforme a bousculé le marché.

Bonne lecture… réelle.

François Tonic



Cédric PRUDENT
Référant réalité virtuelle et augmentée
Expert Web/Javascript
Rédacteur et youtuber
Conserto : ESN conseils et services
ETR : site d'actualités VR/AR

La "réalité virtuelle" ?
De quoi parle-t-on ?

Oculus Rift en action !

sentir physiquement le son

Sensorama

Pour faire au plus simple nous pouvons mettre derrière le terme "réalité virtuelle" l'ensemble des technologies permettant une immersion dans un univers - je vous le donne en mille - virtuel. Car oui il s'agit bel et bien de cela : bluffer le cerveau afin de lui faire croire au maximum sa présence dans un espace purement numérique. Il est donc facile d'imaginer que le but ultime sera de tromper nos 5 sens afin de rendre l'illusion parfaite. A ce stade, nous sommes capables d'en contrôler 3 avec les appareils disponibles sur le marché grand public :

- l'ouïe, le plus facile, par le biais de simples écouteurs,
- le toucher par le biais de retours haptiques dans une paire de gants, une manette ou un contrôleur,
- et surtout la vue par la biais d'un casque ou masque dont le principe de base est très simple : approcher un ou des écrans le plus possible des yeux et par un système de lentilles, principalement de type "Fresnel", pour venir corriger la netteté de l'image.

Oui car sans lentilles vous ne verriez que du flou et encore du flou ; l'oeil humain ne peut faire la "mise au point" ou le focus sur un objet trop proche. Le tout est couplé à des gyroscopes et accéléromètres pour la rotation de la tête et un système de suivi de positions pour le reste du corps. Nous voilà parrés!

Un rêve pas si jeune que ça

Effectivement la genèse de cette technologie remonte à l'année 1956, date à laquelle Morton Heilig, considéré comme le père de la réalité virtuelle, offre au monde son "Sensorama". Il s'agit en fait d'un simulateur extrêmement imposant mais qui permettait déjà à l'époque de projeter un film en 3D synchronisé avec la libération d'odeurs non synthétisées comme le gazon par exemple, un son stéréo ainsi que des vibrations et vent dans les cheveux. Rien que ça.

Depuis moult chercheurs, de la Nasa dans les années 80 aux sociétés de multimédia par la suite n'ont eu de cesse de faire évoluer le concept notamment grâce à la miniaturisation des composants électroniques.

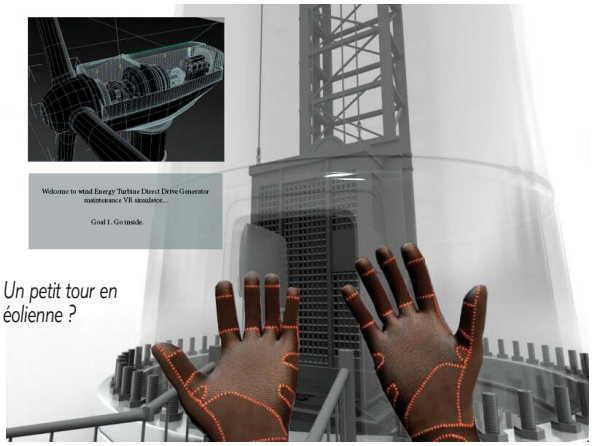
C'est en 2012 qu'un passionné, Palmer Luckey, crée prototypes sur prototypes, fonde son propre forum et acquiert une forte visibilité à tel point qu'un autre passionné et accessoirement une légende dans le monde du jeu vidéo, John Carmack, finit par s'y inscrire et échanger avec lui. C'est donc fort de ce soutien qu'un Kickstarter se prépare, Kickstarter qui finira avec 947% de l'objectif atteint pour un total de 2,4 millions de dollars et 7 000 casques vendus. Il n'en fallait pas plus pour finir de décider tous les constructeurs du monde de l'intérêt de la réalité virtuelle. Cependant un problème de taille les attendait : l'inconfort pour une part non négligeable de la population.

La VR rend-elle malade ?

Si vous vous intéressez au sujet, vous êtes forcément tombés sur des articles traitant du fameux "motion sickness" ou littéralement "le mal des transports" et dont le terme scientifique est : cinétose.

Ce problème majeur en réalité virtuelle est principalement du à deux choses :

- une latence supérieure à 50 millisecondes perturbant la boucle sensori-motrice entraînant des perturbations neuro psychologiques et nausées.
- des décalages de perception entre ce qui est



Un petit tour en éolienne ?

Combo Leap Motion DK2 de chez Oculus

Beaucoup mieux en sans fils avec TPCast

GearVR un Cardboard haut de gamme

Traquer n'importe quel objet réel dans le monde virtuelle

HTC Vive Pro

vécu dans le casque et la réalité. Typiquement dans le cadre de casques sans positional tracking, si vous avancez d'un mètre cela n'aura aucun impact sur votre position au sein de l'environnement virtuel et créera donc un conflit entre les sens pouvant provoquer des nausées pour les plus sensibles d'entre nous.

En tout cas n'ayez crainte, sachez que le seuil de "résistance" à la cinétose est extrêmement variable d'un individu à l'autre, et que, tout comme le mal des transports, une exposition légère et récurrente viendra à bout de toutes sensations désagréables. Seule une infime partie de la population ne pourra se désensibiliser et devra attendre des solutions actuellement en cours de recherche comme des perturbateurs d'oreille interne.

Effet de grille, champ de vision, taux de rafraîchissement, et résolution…

Ce sont là des caractéristiques extrêmement importantes lorsque l'on veut pouvoir choisir au mieux son casque de réalité virtuelle. Commençons par l'effet de grille ou "screen door effect", il s'agit en fait de l'écart entre les pixels qui est rendu visible par le fait de rapprocher l'écran des yeux. Celui-ci peut être atténué par une meilleure résolution d'écran. A l'heure actuelle sur le marché la résolution moyenne des casques est de 1080 x 1200 pixels pour chaque œil, soit 2160 x 1200 pixels en tout avec un taux de rafraîchissement de 90Hz mais elle tend à augmenter rapidement car le HTC Vive pro (casque situé en milieu de génération) rejoint le casque WindowsMR (Windows Mixed Reality) de Samsung et passe à 2880 x 1600. Le tout sur un champ de vision de 110° ce qui correspond à un masque de plongée.

Les casques considérés comme de nouvelle génération auront des résolutions allant jusqu'à 8K, soit un écran 4K par oeil et un FOV de 220° donc à peu de chose près le champ de vision naturel de l'Homme avec un taux de rafraîchissement minimum de 75Hz pour avoir une expérience fluide.

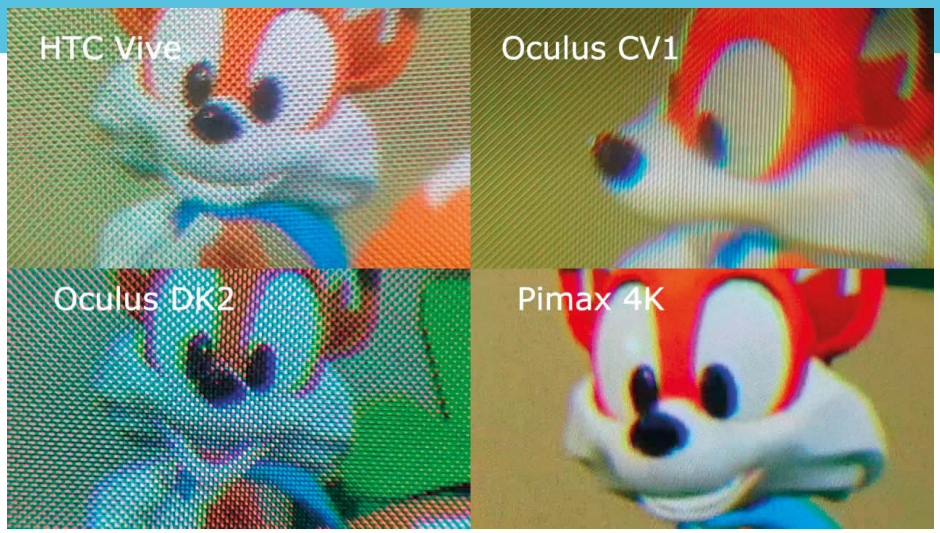
Avec ou sans positional tracking ?

Oui alors on va pouvoir très vite le constater on a fait un bon bout de chemin depuis 2013 et les débuts de la société Oculus avec le DK1 qui est le nom donné à leur premier casque à destination des investisseurs.

Dans la multitude de devices qui sont apparus principalement depuis 2016, on peut y distinguer deux grandes familles.

Il y a tout d'abord les casques qui ne peuvent pas vous positionner dans un environnement virtuel, ils sont dit sans "room scale". Ce sont principalement des casques de type cardboard plus ou moins avancés. Cela commence par de simples assemblages en carton et cela va jusqu'aux casques plus évolués comme le samsung GearVR compatible uniquement avec les téléphones haut de gamme de ladite marque. Il y a aussi le HOMIDO V2 casque de bonne facture compatible avec un très grand nombre de smartphones. Vous n'aurez donc dans ce type de casque que la rotation de la tête ; ce qui en fait des casques principalement conçus pour la visualisation de vidéo 360 et de certaines expériences et/ou jeux dédiés moins immersives. Le manque de positional tracking a un gros désavantage car ses utilisateurs seront plus exposés à la cinétose.

Dans cette catégorie le GearVR possède un avantage, fourni par les capteurs (gyrosco-



Une comparaison des différents "screen door effect"

pe et accéléromètre) supplémentaires placés au sein du casque et plus précis que ceux du smartphone. Ceci ne vous permettra pas pour autant de les utiliser pour du room scale mais rend le tracking vraiment précis. C'est un très bon support si tant est que vous ne soyez pas rebutés par le fait d'utiliser le store d'application de Oculus et leur politique de diffusion. Il est aussi bon de noter que l'Oculus GO, casque autonome, devrait sortir début Mai de cette année et deviendra en théorie la nouvelle référence en matière de casque autonome sans "room scale".

La deuxième famille est donc la famille des casques qui permettra à l'utilisateur de se mouvoir dans le virtuel à la même échelle que dans le réel, ce qui rend tangible l'univers 3D. Seul un très faible pourcentage de personnes seront impactées par le "motion sickness" avec ce type de système car le corps se déplace dans le réel également ne créant ainsi aucun conflit entre les sens.

Parmi la multitude de casques de ce type, le HTC Vive ou Vive Pro prend la place de "leader".

Enfin, plus spécifiquement, vous avez la technologie de tracking laser submillimétrique avec les balises externes nommées LightHouses et développées par Valve, célèbre studio créé par Gabe Newell également fondateur de Steam.

Quand à l'Oculus Rift, il restera un très bon rapport qualité/prix avec son tracking infrarouge qui n'est pas infaillible notamment avec un très faible éclairage mais bien rattrapé par des algorithmes de prédiction efficaces.

Microsoft, apparu sur le marché un peu plus tard, offre quant à lui une approche complètement différente en incluant directement le tracking au sein du casque. Grâce à deux caméras placées à l'avant du casque, fini les balises à calibrer pendant 5 à 10 minutes chez le client afin de faire une démo… La contrepartie se situe principalement au niveau du tracking des contrôleurs. En effet ils doivent impérativement se trouver dans le rayon des caméras pour être détectés. En dehors, seuls les gyroscopes seront fonctionnels, ce qui peut poser des problèmes si votre application doit vous demander d'aller récupérer un élément situé dans votre dos.

Cette famille s'agrandit aussi au mois de Juin avec la société chinoise Pimax, kickstartée et financée à hauteur de 15 Millions d'euros fin 2016, avec le casque Pimax8K basé sur le système de tracking de Valve tout comme les casques HTC.

Tous possèdent des contrôleurs associés où Oculus brille par son "touch" qui possède un système de détection de proximité des doigts et se trouve du coup le plus intuitif à utiliser.

Côté marché il est bon de savoir que le PSVR (casque créé par Sony pour la Playstation 4) est le leader en matière de vente (quasiment 2 millions de vente sur 2017) suivi par les casques pour smartphones. Vient ensuite l'oculus Rift puis le HTC Vive plus coûteux mais qui vous offriront les meilleures des expériences grand public à ce jour.

Obligation d'être relié à un ordinateur ?

Une des évolutions majeures de la réalité virtuelle se situe aussi dans le sans fil. Pour ce faire les constructeurs n'ont que deux possibilités : soit les calculs sont faits directement au sein du casque, il est dit "autonome", soit c'est un émetteur/récepteur WIFI qui va permettre de streamer le flux vidéo du casque au PC ainsi que les informations de positionnement.

Commençons par les casques autonomes, le principe ne sera pas sans rappeler le système de positionnement des casques Microsoft car il s'agit de la même technique : à savoir 2 caméras placées en frontal venant prendre des captures de l'environnement réel afin de connaître par triangulation votre position et calculer vos mouvements. Mais là, mauvaise nouvelle pour l'instant car ils ne sont pas encore accessibles en France voire pas encore sortis. Toutefois 3 casques sortiront en 2018 :

- le HTC Vive Focus : sorti pour l'instant uniquement en Chine il est équipé d'un seul écran AMOLED d'une résolution 3K soit 2880 x 1600 (identique au HTC Vive Pro), équipé d'un processeur Snapdragon 835, le tout pour un rafraîchissement de 75Hz et 110° de FOV (field of view) ce qui est dans la moyenne des casques actuels. Point important le contrôleur de ce casque ne gère seulement que 3 degrés de liberté ce qui signifie qu'il fera plus office de télécommande…
- Le LENOVO Mirage quant à lui se compose d'un écran LCD ce qui rendra une image beaucoup plus terne que le Vive Focus et dont la résolution sera de 2560 x 1440. il est également doté d'un Snapdragon 835 et d'un FOV de 110°. Son contrôleur ne fera également qu'office de télécommande et malheureusement n'espérez pas avec ce casque bénéficier d'un "room scale" de plus de 2 mètres ; en dépassant cette distance, le casque vous demandera gentiment de revenir à votre position d'origine…
- Le PICO Néo quant à lui n'est armé que d'un Snapdragon 820 (le même que les Samsung S7) et possède une résolution 3K répartie sur 2 écrans. L'originalité de ce casque est que le room scale proposé ici ne marchera qu'à l'aide d'accéléromètres, magnétomètre et gyroscope ce qui me laisse perplexe quant à sa finesse de tracking...

Regardons maintenant du côté de la transmission sans fil, là où pas mal de startups et de gros constructeurs également se livrent à une course de vitesse. Notamment INTEL et TPCast avec des résultats identiques quant au remplacement du passage par câbles : Ils se basent sur une nouvelle technologie, le WiGig, venant diminuer la latence du streaming afin de la réduire en dessous des 7ms. Élément intéressant concernant le design du modèle de INTEL, les "cornes" permettent de faire du load balancing afin de répartir la charge des données reçues mais également de mieux gérer la position de l'utilisateur pour rediriger le signal et donc éviter pertes et latences.

Accessoires pour encore plus d'immersion !

Nous avons essayé de vous faire le tri parmi les innombrables accessoires disponibles et surtout de les catégoriser en trois grandes sections avec en premier les accessoires de tracking :

- Avec les Vive trackers (attention uniquement fonctionnels avec les casques de la marque HTC mais sachez qu'il existe des moyens de contourner cette restriction malgré tout). Ils sont

Le cardboard "cheap"

ILe futur sans fils WiGiG

extrêmement intéressants notamment sur le plan professionnel car ils vont permettre de traquer n'importe quel élément réel dans l'univers virtuel. Le principe est simple vous allez fixer le tracker sur un objet, prenons comme exemple une application de formation initiale sur un poste de chirurgie ; il vous suffira de modéliser l'élément en 3D, de placer le tracker précisément, et hop, voilà votre ustensile à l'échelle 1:1 dans votre scène et traqué en temps réel.

- Le Leap Motion quant à lui va vous permettre de traquer vos doigts en temps réel afin de pouvoir les utiliser comme contrôleur. L'implémentation est solide et efficace sur les moteurs comme UNITY et UNREAL Engine, le seul problème reste l'angle de tracking qui vous imposera de laisser vos mains bien en face de vous pour garantir que le tracking ne décroche pas.
- Le 3drudder également est un accessoire qui vous permettra de vous mouvoir grâce à vos pieds, ce qui peut enlever une partie du "motion sickness" car votre corps reste en mouvement et surtout vous permet de rester assis sans briser l'immersion de l'expérienceur.

Les accessoires pour un supplément d'immersion :

- Le Subpack utilisé dans le monde professionnel de la musique. Il apporte sous sa forme "sac à dos" un véritable plus. Le principe de base est très simple, il s'agit d'une sorte de caisson de basses réagissant aux fréquences basses et permet de ressentir physiquement le son. Le gros avantage de ce type de produit est donc d'être compatible avec toutes les applications sans avoir à faire aucun développement.
- Les fauteuils DBOX. Ils sont de plus en plus présents dans nos salles de cinéma préférées et peuvent désormais intégrer facilement nos salons.

Les accessoires de retours haptiques :

Ile déplacement en VR assis et avec les pieds

Énormément de prototypes commencent à affluer dans cette catégorie basée sur des technologies parfois inattendues :

- UltraHaptics vous propose un DevKit permettant de créer la sensation de toucher grâce au contrôle des ultrasons ;
- Teslasuit vous proposeront des combinaisons vous permettant de ressentir la chaleur/fraîcheur par le biais de l'effet Peltier mais également la sensation de toucher grâce à l'électrostimulation ;
- Microsoft a publié récemment un prototype plus évolué codename Claw permettant avec un système de retour de force de pouvoir distinguer des formes.

Technologie uniquement destinée aux jeux ?

On a tendance à cataloguer la VR, à tort, comme étant uniquement destinée aux jeux vidéo ou d'une façon plus générale au secteur du divertissement. Ce qui à mon sens est une erreur car certes c'est un cas d'usage qui lui sied à merveille mais ce nouveau médium peut apporter tellement plus.

Tout d'abord le "serious gaming" autrement dit la formation, qu'elle soit initiale ou continue est un sujet qui marche très fort en ce moment avec comme exemple concret la formation de personnel de maintenance et plus particulièrement maintenance pour éoliennes. Initialement un formateur n'avait d'autre choix que de montrer en réel aux futurs techniciens les éléments qui composent une éolienne et par manque de place dans celle-ci les sessions de formation ne pouvaient contenir qu'un nombre extrêmement limité d'apprenants. Maintenant imaginez pouvoir transcrire l'intérieur d'une éolienne… Vous avez alors une formation initiale sans risque de chute ou autre, et pouvant potentiellement contenir un nombre indéfini de personnes, le tout dispensé par un seul formateur.

Ou encore dans le médical où grâce à des simulateurs nos futurs chirurgiens s'exercent et apprennent les différentes étapes de réalisation d'opérations spécifiques qui auparavant ne pouvaient être réalisées que sur des cadavres limitant énormément le nombre d'essais.

Le domaine de l'éducation va également être fortement impacté car il est démontré scientifiquement que l'immersion engendre une attention plus forte et par extension une mémorisation plus efficace.

Sachez tout de même qu'il est difficile (pour le moment en tout cas) de trouver des subventions même pour des projets du domaine médical. Je vous conseille de vous rapprocher de consortiums déjà en place selon le secteur que vous aurez choisi afin d'être le mieux informé possible sur les possibilités de financement ou de partir directement sur un développement en fonds propres.

Comme toutes nouvelles technologies, le potentiel ne dépend que de l'imagination des uns et des autres et en cela je ne doute pas que vous trouverez de nombreux cas d'usage.

Quid de l'avenir ?

Comme toutes nouvelles technologies (qui plus est à fort potentiel), l'émulsion canalisée de tous les acteurs majeurs offre une évolution rapide. En effet, en plus d'un standard en préparation nommé openXR auquel tous les plus gros participent, nous ne tarderons pas à bientôt percevoir Internet comme un univers 3D. En effet le WebVR de par les frameworks et solutions qui prolifèrent (babylon.js, A-FRAME…) devrait rapidement s'imposer comme un incontournable dans les prochaines années.

Mais il y a également beaucoup d'avancées du côté du "eyes tracking" avec entre autres la société Tobii qui fournit déjà à l'heure actuelle un SDK et un device plus que convaincants. Cela permet un niveau d'immersion supérieur grâce aux nombreuses interactions indirectes que l'on peut créer en prenant en compte ce que regarde exactement l'utilisateur. Le tracking des pupilles peut également permettre de revoir les systèmes de navigation dans les menus afin de les rendre plus intuitifs. J'ai eu la chance de pouvoir tester leurs démos et je peux affirmer que le confort de navigation dans les interfaces y est vraiment instinctif.

Et pourquoi pas, grâce à la miniaturisation, imaginer nos casques réduits à la taille d'une simple paire de lunettes. Mieux jumeler réalité virtuelle et réalité augmentée … •

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

La réalité virtuelle transforme le Web : WebVR & WebXR

Le futur commence aujourd'hui ! Les réalités augmentée (AR), virtuelle (VR) et la miXte (XR) s'invitent dans les navigateurs pour accéder à des contenus d'un nouveau genre, grâce à de nouvelles API !

Depuis sa création, **Internet se présente sous la forme d'une interface en deux dimensions, regroupant principalement du texte, des images et des vidéos**. Toutefois, grâce à la réalité virtuelle, le web pourrait devenir un véritable monde virtuel. La technologie s'invite un peu plus dans les navigateurs pour accéder à un « internet moderne » à travers de nombreuses API comme la réalité virtuelle, qui va réellement transformer le web que vous connaissez aujourd'hui, à travers de nouveaux standards, afin de lui rendre la navigation plus confortable.

La réalité augmentée : WebAR

Il s'agit d'une technologie qui va intégrer des fonctionnalités de la réalité augmentée dans vos navigateurs. L'intérêt de cette technologie est de permettre aux designers, médias et autres créatifs de créer des objets virtuels en 3 dimensions (3D) et de les intégrer directement aux sites web ou sur son mobile. Ainsi les utilisateurs peuvent les superposer dans leur environnement réel.

Par exemple :

Dans une utilisation personnelle, de placer un modèle 3D dans une pièce d'habitation et le faire tourner sous toutes ses coutures pour voir le rendu avant un achat final. Cet usage peut intéresser de nombreux secteurs d'activité.

Enfin, la Web AR pourrait être utilisée pour vous accompagner dans la rue pour vous aider à vous déplacer (un peu comme avec les Google Glass, NDLR).

La réalité virtuelle : WebVR

Avec les différents matériels (principalement les casques) de réalité virtuelle et maintenant avec les navigateurs, il y a un nouveau centre d'intérêt. WebVR permet d'accéder à du contenu en réalité virtuelle depuis un navigateur.

Par exemple :

Les studios de jeux, des vidéos, des contenus interactifs seront les premiers secteurs intéressés par le WebVR.

La miXte réalité : WebXR

Il s'agit d'une technologie de réalité mixte sur le web, c'est-à-dire la possibilité d'utiliser aussi bien la réalité virtuelle, que de la réalité augmentée.
L'intérêt va permettre de mieux répondre à la fragmentation du marché et résoudre les problèmes de compatibilité, avec une couche API WebXR. De cette façon, il va être plus facile de réaliser des sites Web cross-browsers (multi-navigateurs). Il existe actuellement de nombreux frameworks comme ARCode, ARKit, Hololens, aFrame... Ce format est compatible avec les nouvelles générations de matériels et d'OS.

Cas pratiques : A-FRAME

A-Frame est un framework open source & libre, maintenu par la fondation Mozilla. Il permet de créer de la 3D avec des vues en 360 degrés, des pages web et des scènes en réalité virtuelle (webVR) grâce à des balises HTML. L'utilisation de ce framework permet aux développeurs web de créer facilement des scènes de réalité virtuelle. Il fonctionne à 100 % dans le navigateur (desktop et mobile) : aucune extension ou app supplémentaire n'est nécessaire. Les performances du matériel et une bonne connexion pèseront sur la rapidité d'affichage.

Fonctionnalités principales

Avant d'utiliser le framework A-Frame, il est nécessaire d'appeler une librairie dans la balise <head> : l'entête de votre fichier HTML.

```
<script src="aframe.min.js"></script>
```

L'opération suivante sera déclarée dans le body de votre page HTML. Pour cela, vous utilisez les balises <a-scene></a-scene> pour déclarer une scène dans votre page HTML
Nous vous proposons de réaliser ces 3 concepts dans une page web à partir des exemples proposés par le site aframe.io

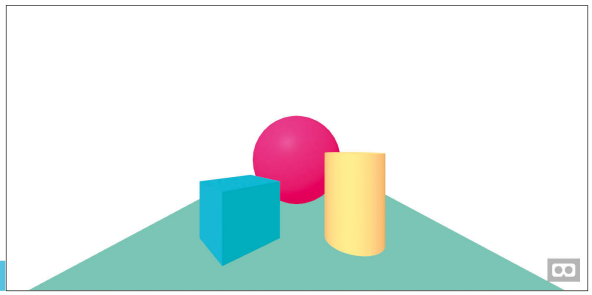
La réalité virtuelle : WebVR

Nous vous proposons 2 exemples pour vous montrer quelques possibilités de WebVR.

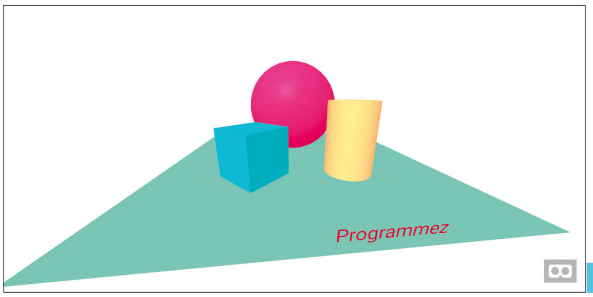
Exemple 1 : des objets dans un espace

Commençons par créer 3 formes géométriques en 3D dans une scène : 1

```
<a-scene>
 <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
 <a-box position="-1 0.5 -3" rotation="0 45 0" width="1" height="1" depth="1" color
="#4CC3D9"></a-box>
 <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D">
</a-cylinder>
 <a-plane position="0 0 -4" rotation="-90 0 0" width="6" height="6" color="#7B
C8A4"></a-plane>
</a-scene>
```

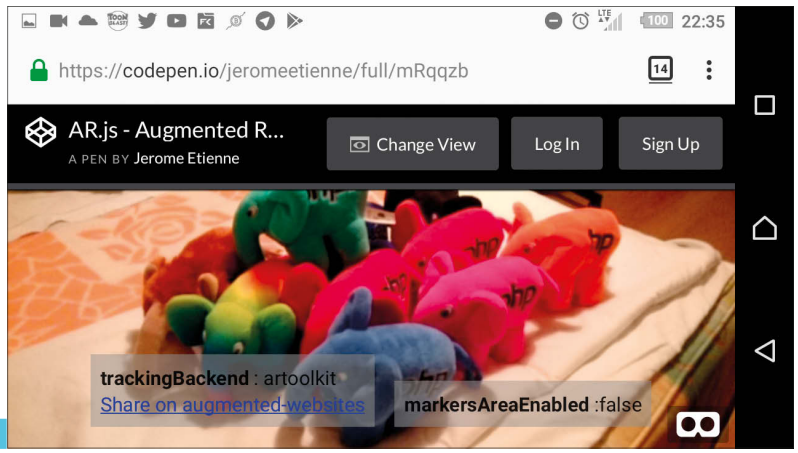


1



2

3



4

Les balises suivantes représentent des formes géographiques :

- <a-sphere> une sphère ,
- <a-box> un carré ,
- <a-cynlinder> un cylindre,
- <a-plane> définition d'une surface plane à l'aide du composant de géométrie.

Chaque balise contient des attributs, comme :

- une position : X Y Z,
- une rotation,
- une dimension : largeur / hauteur,
- une couleur,
- etc.

Nous en profitons pour ajouter le texte 'Programmez' qui se décompose de la manière suivante :

```
<a-text position="0 0.25 -1" height="10" color="#FF0000" value="Programmez">
</a-text>
```

- <a-text> enveloppe le composant texte

La position est calculée par rapport à la balise <a-plane> pour le positionnement au premier plan et en respectant la perspective.
Pour voir le résultat, il suffit de déplacer la scène comme le montre l'image 2.

Exemple 2 : le visuel

Le visuel est souvent représenté par une image, un paysage ou tout autre décor. C'est pourquoi nous allons réaliser un 360°, de la manière suivante :

```
<a-sky src="panorama.jpg" rotation="0 -31 0"></a-sky>
```

<a-sky> affiche une image en fond d'écran
Pour réaliser un 360°, il est préférable de charger une image de très grande taille, comme le montre l'image 3.

La réalité augmentée : WebAR

Pour utiliser la réalité augmentée avec A-FRAME, nous utiliserons la librairie AR.js pour communiquer avec les différents périphériques de votre appareil et améliorer l'utilisation sur internet.
La librairie AR contrôle le déplacement de la caméra ou la webcam en l'associant avec le framework A-FRAME, l'affichage s'effectuera dans votre navigateur.
Pour appeler celle-ci, nous ajoutons la ligne suivante dans le <head> de notre page web.

```
<script src="https://jeromeetienne.github.io/AR.js/aframe/build/aframe-ar.js"></script>
```

Exemple de base

Pour que notre caméra soit disponible dans notre scène, nous complétons la balise <a-scene> :

```
<a-scene embedded arjs='sourceType: webcam;'>
```

La gestion du mouvement de la caméra dans la scène, s'effectuera comme ceci :

```
<a-marker-camera preset='hiro'></a-marker-camera>
```

Cette balise ajoute un pointeur 'hiro' dans notre scène pour que la réalité augmentée soit prise en compte dans le navigateur afin d'obtenir le résultat de l'image. 4

Le code se résumera à ceci :

```
<a-scene embedded arjs='sourceType: webcam;'>
  <a-box position='0 0.5 0' material='opacity: 0.5;'></a-box>
  <a-marker-camera preset='hiro'></a-marker-camera>
</a-scene>
```

Dans cette scène, la caméra est déplacée par AR.js, et l'origine de votre scène est au centre du marqueur. Si vous voulez ajouter de nouveaux objets, il suffit de l'ajouter près de la <a-box> .

Exemple avancé

Il est possible d'utiliser des librairies complémentaires comme three.js ou des balises du framework, de la façon suivante :

```
<a-text position="0 0.25 -1" height="10" color="#FF0000" value="Programmez">
</a-text>
```

pour obtenir le résultat suivant : 5

La miXte réalité : WebXR

Comme il s'agit d'une réalité mixte, nous utilisons aframe que nous ferons évoluer avec des librairies supplémentaires.

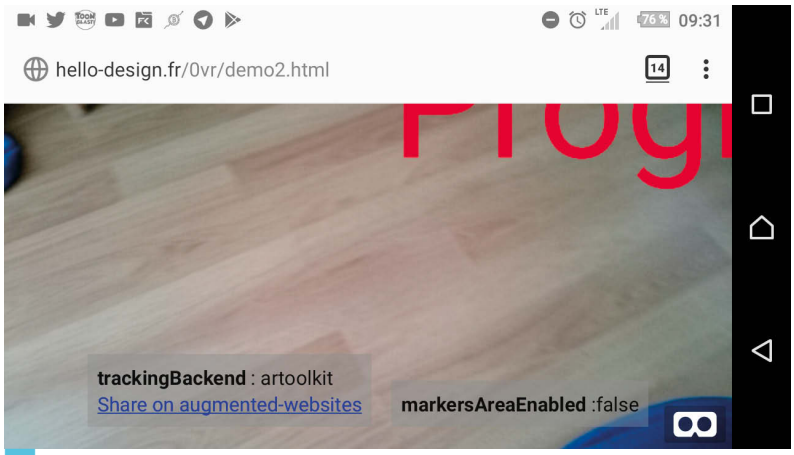
```
<script src='aframe-master.js'></script>
<script src='three.xr.js'></script>
<script src='aframe-xr.js'></script>
<script src='hit-test.js'></script>
```

Le principe d'utilisation reste identique à ce que nous avons déjà vu. Tout d'abord avec la déclaration de la scène et à la déclaration de méthodes et de classe

```
<a-scene hit-test></a-scene>
```

Pour interagir avec notre scène, nous utiliserons des objets dans notre espace de réalité augmentée.

```
<script>
  var scene = AFRAME.scenes[0];
  var newObject = function(data) {
    var entity = data.detail;
    entity.setAttribute('geometry', {
      primitive: 'box',
      width: 0.1,
      height: 0.1,
      depth: 0.1
    });
    entity.setAttribute('material', {
      shader: 'standard',
      transparent: true,
      opacity: 0.45,
      fog: false,
      color: '#ffcc00'
    });
```



5

```
    scene.appendChild(entity);
  }
  scene.addEventListener('newAnchoredEntity', newObject);
</script>
```

Le script montre la possibilité de manipuler les objets 3D dans un espace réel. Le code source original est disponible : https://github.com/mozilla/aframe-xr

Les outils Cross platform

Pour l'heure, le marché de la réalité virtuelle demeure très fragmenté. Tous les mois sont commercialisés un ou plusieurs nouveaux casques avec leur propre écosystème, et des boutiques d'applications.

Toutefois, WebVR permet une compatibilité multi-navigateurs avec tous les casques du marché, sans avoir à télécharger une application supplémentaire. A cette occasion, des navigateurs dédiés commencent à apparaitre comme 'Firefox Reality', prévu dans les prochaines semaines. Ceci permet aux fabricants d'adapter plus facilement le navigateur à leurs appareils.

Du côté des outils de développement, ils sont identiques à ceux que vous utilisez déjà.

Conclusion

Google a apporté le support de WebVR à Chrome sur certains téléphones Android compatibles Daydream en février dernier. Firefox est devenu le premier navigateur de bureau à supporter officiellement la norme WebVR.

Mais le futur devrait laisser la place au WebXR. •

Liens

Pour aller plus loin :

Demo : https://aframe.io

Documentation : https://aframe.io/docs/

A-frame VR : https://github.com/aframevr/aframe

Mozilla Reality : https://mixedreality.mozilla.org

WebXR : https://github.com/mozilla/aframe-xr

Blog mozvr : https://blog.mozvr.com

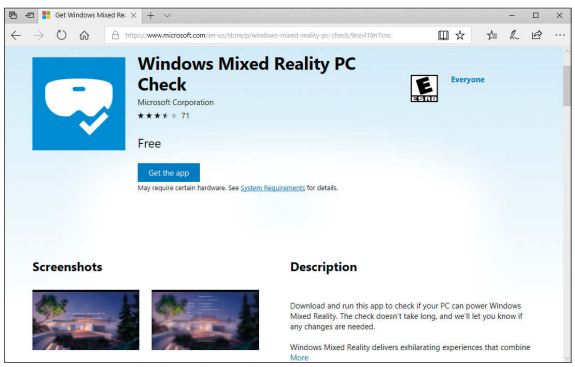
Sebastien Bovo
Windows AppConsult Engineer chez Microsoft. En charge d'accompagner les partenaires et développeurs sur la plateforme Windows Mixed Reality Immersive et Holographique pour la publication sur le Store Microsoft.
sbovo@microsoft.com | twitter.com/sbovo/

Créer une application Windows Mixed Reality à partir d'une feuille blanche

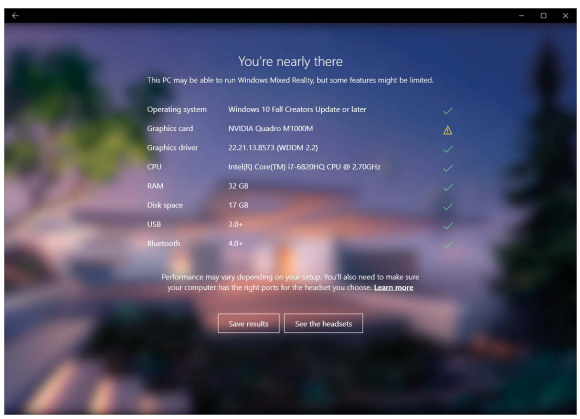
Depuis la version Windows 10 Fall Creators Update d'octobre 2017, tout PC suffisamment puissant et équipé d'un casque « Windows Mixed Reality » sera capable de vous transporter dans le monde virtuel ! Génial ! Mais comment savoir si un PC est compatible Windows Mixed Reality ? Quelles sont les étapes pour développer une application à partir de 0 ? Quels outils utiliser ?

La configuration matérielle

Nous l'avons évoqué : depuis octobre 2017, Windows vous permet d'exécuter des applications VR natives Mixed Reality. Quel matériel faut-il ? Si vous avez déjà un PC, le plus simple est de tester sa compatibilité avec l'application du Store Microsoft appelée Windows Mixed Reality PC Check.



Cette application fait son travail : un bouton (I agree) et quelques secondes après, les résultats des tests de compatibilité s'affichent.



Si vous n'avez pas un PC compatible ou si vous souhaitez en acheter un, voici un aperçu du minimum demandé :

- un processeur Intel Core i5 (7ième génération mobile) double-cœur,
- 8Go de RAM DDR3,
- une carte graphique Intel HD Graphics 620 compatible DX-12 ou NVIDIA MX150/965M compatible DX-12,
- un port HDMI 1.4 ou DisplayPort 1.2,
- un port USB 3.0
- Bluetooth 4.0.

Pour obtenir tous les détails de la configuration minimale et optimale, recherchez "Windows Mixed Reality minimum PC hardware" sur https://docs.microsoft.com/.

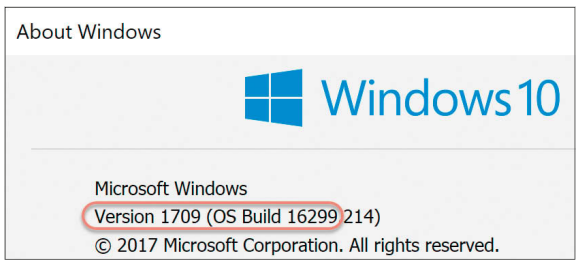
Enfin en possession d'un PC compatible, vous avez besoin d'un casque Windows Mixed Reality. Vous pouvez effectuer l'achat chez un commerçant près de chez vous ou commander sur le Store Microsoft - https://store.microsoft.com/. Les casques disponibles en France sont ceux d'ACER, d'ASUS, DELL et LENOVO. Tous sont livrés avec les contrôleurs et partagent les mêmes caractéristiques techniques. A vous de voir lequel vous trouvez le plus confortable ou de vous décider en fonction du prix.

La technologie et les outils

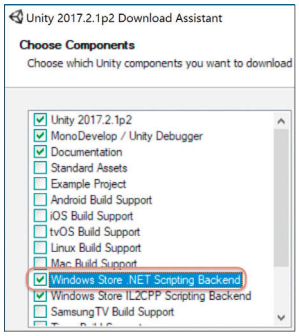
Vous avez donc maintenant le PC et un casque Windows Mixed Reality, nous pouvons aborder la technique ! Une application Mixed Reality utilise les APIs du SDK Windows. Deux choix s'offrent à nous. Le premier est d'utiliser ces APIs directement par code en C++. Nous sommes dans ce cas au plus près du métal et donc avec le maximum de performance mais il faut tout coder : le moteur 3D, la gestion des contrôleurs, etc. Ce n'est pas la méthode la plus rapide pour développer une application, mais certains scenarios demandent ce type de programmation. L'autre possibilité consiste à tirer parti d'un outil de création de jeux comme Unity3D. Nous sommes ainsi capables de créer nos scènes 3D, lumières, caméras, effets, et d'en scripter les interactions avec du code C#.

Partons sur l'utilisation d'Unity3D et voici donc les versions recommandées à date :

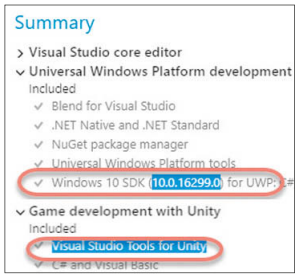
- Windows Fall Creators Update (Version 1709 - OS Build 16299.15) ou supérieur



- Unity3D 2017.2.1p2 (Cette version est la plus stable actuellement pour la plateforme Mixed Reality). Assurez-vous d'avoir le composant **Windows Store .NET Scripting Backend** coché lors de l'installation.



- Visual Studio 2017 Version 15.5 ou supérieur. Dans l'installeur Visual Studio, vérifiez que vous avez les composants **Windows 10 SDK 10.0.16299** et **Visual Studio Tools for Unity** sélectionnés.



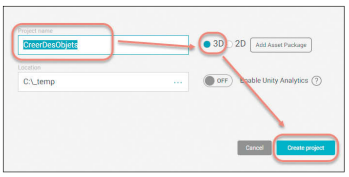
Tous les détails et les liens de téléchargement sont disponible sur aka.ms/StartMixedReality.

Le projet Unity

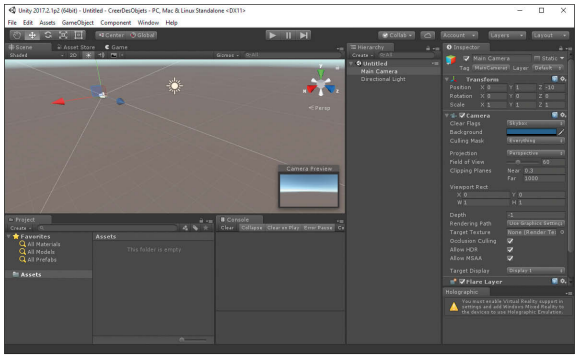
Nous avons dit "A partir d'une feuille blanche" ? Allons-y !

- Lancer Unity3D ; l'écran d'accueil vous permet d'ouvrir un projet existant ou d'en démarrer un nouveau, choisir d'en créer un nouveau en cliquant sur **New**.

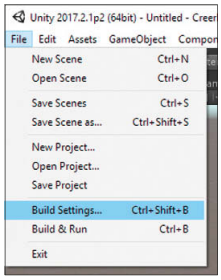
- Donner un nom au projet comme "CreerDesObjets", s'assurer que **3D** est coché ; cliquer sur **Create Project**.



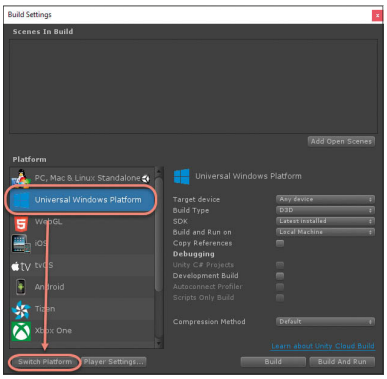
- Une fois le projet créé, l'interface d'Unity s'affiche.



- Afin de cibler la plateforme Windows Mixed Reality, utiliser le menu **File** puis **Build Settings**...



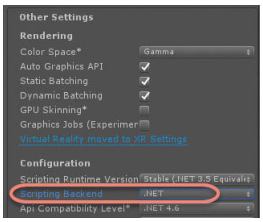
- Choisir **Universal Windows Platform** puis cliquer sur **Switch Platform**.



Cette action a pour effet d'indiquer à Unity que le projet est à destination de la plateforme Windows Mixed Reality. Vous avez ici la confirmation qu'une application Mixed Reality est, avant tout, une application Universal Windows Platform (UWP).

- Les derniers paramétrages nécessaires consistent à préciser que nous supportons la VR et que le scripting se fait en C#. Pour ce faire, toujours dans l'écran **Build Settings**, cliquer sur **Player Settings**.

- Le fenêtre **Inspector** affiche maintenant les options relatives à l'application. Cliquer sur **XR Settings** et cocher **Virtual Reality Supported**. Puis dans **Other Settings**, sélectionner **.NET** comme **Scripting Backend**.

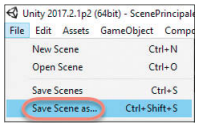


Votre application est prête ! Mais vide

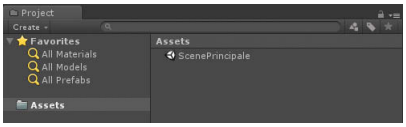
Un exemple simple

Nous partons donc du nouveau projet vide que nous avons configuré pour Windows Mixed Reality. Commençons par sauvegarder la scène courante.

• Cliquer sur le menu **File** puis **Save Scene as…**



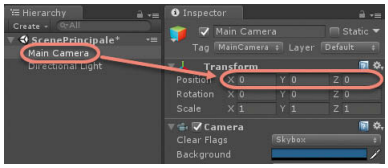
• Choisir "ScenePrincipale" pour le nom de la scène. Le fichier sauvegardé se retrouve dans le répertoire **Assets** du projet comme absolument tout ce qui va constituer votre projet (objets 3D, textures, scripts).



Positionnons ensuite la caméra à l'origine de la scène (x=0, y=0, z=0). Elle représente notre personne ; sa position et rotation seront celles de notre casque Mixed Reality.

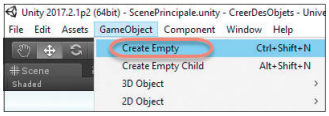
• Cliquer sur l'objet **Main Camera** dans la fenêtre **Hierarchy** ; cela a pour effet de sélectionner l'objet et la fenêtre **Inspector** en affiche maintenant ses propriétés,

• Dans la fenêtre **Inspector** et le groupe **Transform**, changer les valeurs de x, y et z pour avoir "0" pour chacune des coordonnées de position.



Maintenant, ajoutons un script pour intercepter l'action du bouton principal des contrôleurs (il s'agit de la gâchette placée sous le contrôleur ; c'est le bouton appuyé par l'index). Nous pouvons attacher ce script sur un objet existant comme la caméra ou créer un GameObjet vide qui va servir de conteneur. Utilisons la seconde méthode pour une meilleure lisibilité du projet.

• Dans le menu **GameObject,** cliquer sur **Create Empty.**

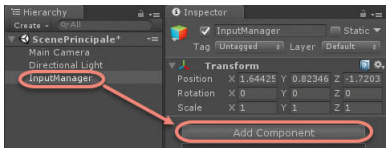


Dans la fenêtre **Hierarchy**, cliquer sur l'objet nouvellement créé appelé "GameObject",

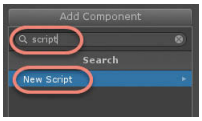
• Cliquer une seconde fois pour en modifier le nom (ou appuyer sur F2).

• Nommer le GameObject "InputManager",

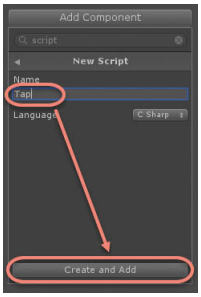
• Toujours avec le GameObject **InputManager** sélectionné dans la fenêtre **Hierarchy**, cliquer sur le bouton **Add Component** de la fenêtre **Inspector.**



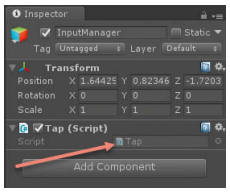
• Dans la zone de recherche, taper "Script" et cliquer sur **New Script.**



• Entrer le nom "Tap" et cliquer sur **Create and Add.**



• Nous avons donc maintenant un nouveau script associé à l'objet **InputManager**. Pour le modifier double-cliquer sur le nom du script.



Visual Studio est l'éditeur de code par défaut pour Unity. Le script ouvert n'est pas vide, voici le contenu par défaut d'un script Unity :

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

public class Tap : MonoBehaviour {

  // Use this for initialization
  void Start () {

  }

  // Update is called once per frame
  void Update () {

  }
}
```

Les deux commentaires sont explicites. Nous voyons ici que nous sommes bien dans le contexte des jeux vidéo avec du code qui va s'exécuter une seule fois (Start) et à chaque frame (Update) c'est-à-dire au moins 60 et 90 fois par seconde si le PC le permet.

Nous allons donc enfin écrire quelques lignes de C# pour intercepter le bouton principal des contrôleurs. Nous avons à notre disposition une classe **GestureRecognizer**. Cette classe permet de reconnaître les gestes HoloLens, les boutons de manettes Xbox ou de contrôleurs. Voici comment l'utiliser :

• Utiliser l'espace de noms contenant cette classe.

```
using UnityEngine.XR.WSA.Input;
```

• Ajouter un membre privé de type **GestureRecognizer.**

```
private GestureRecognizer recognizer;
```

• Dans la méthode **Start**(), instancier le membre **recognizer**.

```
recognizer = new GestureRecognizer();
```

Indiquer ensuite que nous cherchons à intercepter le geste de type

Tap (Comme nous l'avons dit, la classe est suffisamment haut niveau pour regrouper aussi bien le **AirTap** d'HoloLens avec l'appui sur le bouton principal du contrôleur Mixed Reality). Nous utilisons la méthode **SetRecognizableGestures()**.

```
recognizer.SetRecognizableGestures(GestureSettings.Tap);
```

- Afin d'être notifié lorsque le Tap est effectué (ou dans notre cas le bouton du contrôleur), nous nous abonnons à l'évènement **Tapped** et nous démarrons la capture des gestes.

```
recognizer.Tapped += GestureRecognizer_Tapped;
recognizer.StartCapturingGestures();
```

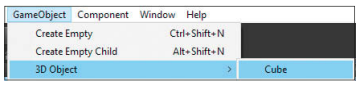
C'est maintenant que nous allons pouvoir créer un peu de matière virtuelle ! Sur chaque pression du bouton du contrôleur, la méthode **GestureRecognizer_Tapped** est exécutée. Voici ce que l'on peut simplement faire :

- Créer un **GameObjet** simple : un cube ;
- Changer sa taille pour x=1, y=0.1, z=1. Nous avons donc une planche de 1m sur 1m (x et z) avec pour épaisseur 10cm (y) ;
- Changer sa position pour que la planche soit en face de soi (x=0), à 2m de hauteur (y=2) et à 4m de distance (z=4);
- Nous indiquons que l'objet est soumis à la gravité (et le moteur Unity fera le reste).

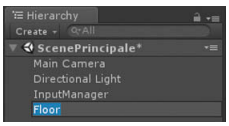
```
private void GestureRecognizer_Tapped(TappedEventArgs obj)
{
    var o = GameObject.CreatePrimitive(PrimitiveType.Cube); //a
    o.transform.localScale = new Vector3(1f, 0.1f, 1f); //b
    o.transform.position = new Vector3(0f, 2f, 4f); //c
    o.AddComponent<Rigidbody>(); //d
}
```

Comme vous pouvez le deviner, à chaque pression du bouton du contrôleur, nous allons créer une planche. Parce que nous leur avons affecté un **Rigidbody**, ces planches vont tomber dans le vide indéfiniment. Ajoutons donc une surface plane sur laquelle ces planches pourront chuter. Retournons dans le projet Unity :

- Dans le menu **GameObject**, cliquer sur **3D Object** puis **Cube**.

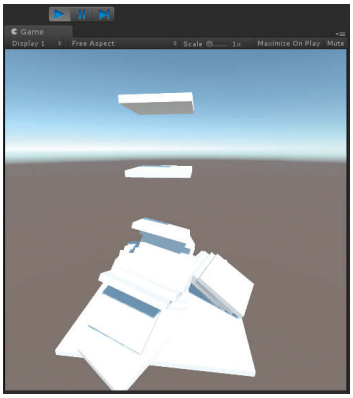


- Dans la fenêtre **Hierachy**, renommer le cube en le sélectionnant puis F2 ; l'appeler "Floor".



- Changer sa position pour qu'il soit face à nous (x=0) à 4m (z=4) et au niveau du sol (y=-1.8) si l'on considère que l'on mesure à peu près 1m80. Changer ses dimensions pour passer d'un cube à une surface plane d'une largeur et d'une longueur de 3m (x=3, z=3) pour une épaisseur de 10cm (y=0.1).

Nous avons terminé ! Pour tester, cliquer simplement sur le bouton **PLAY** d'Unity  : Vous visualisez le rendu dans la fenêtre **Game** d'Unity mais aussi et surtout dans le casque Windows Mixed Reality !



Voici le code complet du script :

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.XR.WSA.Input;

public class Tap : MonoBehaviour
{

    private GestureRecognizer recognizer;

    // Use this for initialization
    void Start()
    {
        recognizer = new GestureRecognizer();
        recognizer.SetRecognizableGestures(GestureSettings.Tap);
        recognizer.Tapped += GestureRecognizer_Tapped;
        recognizer.StartCapturingGestures();
    }

    private void GestureRecognizer_Tapped(TappedEventArgs obj)
    {
        var o = GameObject.CreatePrimitive(PrimitiveType.Cube); //a
        o.transform.localScale = new Vector3(1f, 0.1f, 1f); //b
        o.transform.position = new Vector3(0f, 2f, 4f); //c
        o.AddComponent<Rigidbody>(); //d
    }

    // Update is called once per frame
    void Update()
    {

    }
}
```

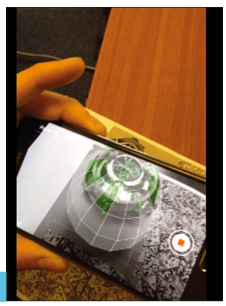
Pour aller plus loin

Nous avons posé les fondations et abordé ici la création d'une application Windows Mixed Reality. Nous verrons prochainement comment aller beaucoup plus vite avec le Mixed Reality Toolkit. Nous traiterons notamment de sujets comme la gestion précise des contrôleurs, la téléportation et nous irons plus loin sur l'utilisation d'Unity. Dernier point, et sûrement l'un des plus importants : rendre votre œuvre publique sur le Store Microsoft ! •

Eric Duport
SQLI Lyon

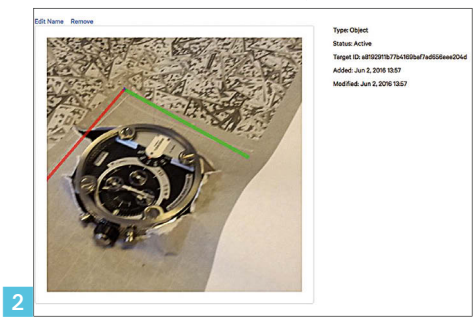
Augmentez la réalité de vos objets 3D avec Vuforia

Du point de vue du développeur mobile comme du mien, une solution telle que Vuforia représente un SDK cross platform riche, très riche, avec des outils efficaces. Parmi ces outils, on trouve une détection de marqueurs (ou marker, image à charger à la compilation de l'application) qui servent de repères pour un objet 3D à scanner. Oui, un objet 3D ! Nombreuses alors sont les possibilités fonctionnelles autour de ce que l'on peut associer à cet objet : tutoriels, guide d'utilisation, supports marketing…



1

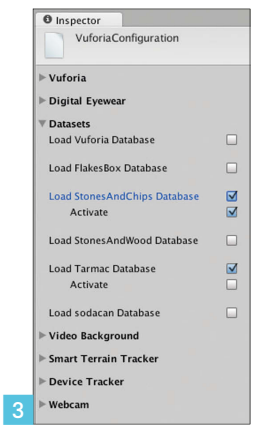
Pour gérer cette reconnaissance d'objet, il faut créer un marqueur (objet 3D compatible). Afin de créer ces objets, Vuforia met à sa disposition un exécutable Android appelé Vuforia Object Scanner ainsi qu'une feuille de cible à imprimer. Le principe est alors simple, il suffit de lancer l'APK fourni par Vuforia et de scanner l'objet à intégrer comme marqueur. Cela crée alors un nuage de points qui sera la signature numérique de ce marqueur. La saisie des données consiste à réaliser des «scans» depuis différents points de vue de la scène à numériser. Pour des environnements complexes ou de grande taille, il peut être nécessaire d'effectuer un grand nombre de points de vue (quelques dizaines).



2

Exemple avec une montre : 1

Armé du fichier objet alors généré (d'extension .od), nous aurons à notre disposition, sur l'espace target manager de notre compte Vuforia, la possibilité de créer une base de données qui sera chargée directement depuis le téléphone. Depuis l'interface Target Manager, il est possible d'uploader des fichiers .od. Nous pourrons alors exporter notre base de données afin de la placer dans le dossier assets de notre application.



3

Exemple d'implémentation

Pour ma part j'utilise Unity pour la création d'applications mobiles utilisant de la réalité augmentée. L'implémentation est la suivante :

1 – Créez une base de données dans le gestionnaire de cible qui inclut la cible d'objets que vous souhaitez utiliser : 2

2 – Téléchargez cette base de données sous la forme d'un fichier *.unitypackage.

3 – Importez ce package unity en allant dans Asset –> import Package dans l'éditeur Unity. Ce package sera ajouté dans le dossier Streaming Assets/QCAR.

Pour ajouter et configurer le marqueur d'objet sous forme de GameObject :

1 – Ajoutez un objet de type ARCamera à votre scène dans Unity (menu : GameObject>Vuforia> AR Camera). Supprimez la caméra par défaut de votre scène.

2 – Allez dans la section Databases de la configuration de Unity (menu: Window>Vuforia Configuration) et vérifiez que votre base de données soit activée : 3

3 – Ajoutez un ObjectTarget à votre scène (menu : GameObject>Vuforia>3D Scan), l'objet AR Camera et cet objet devront être au même niveau dans la hiérarchie de la scène. 4

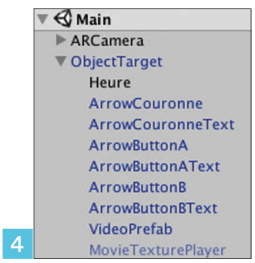
4 – Cliquez sur Object Target Behaviour dans l'inspecteur de l'objet target et sélectionnez le nom de votre base de données à y associer avec votre marqueur. Il ne peut y avoir qu'une seule base de données associée à l'objet target. 5

5 – Mettez les valeurs à zéro pour les positions de cet objet target. 6

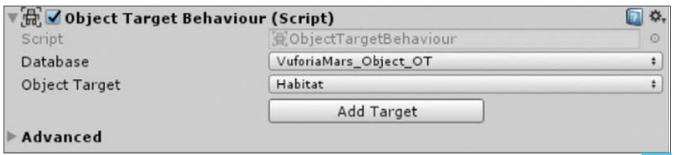
6 – Ajoutez votre contenu (video, objet 3D, textes, images…) qui se superposera à votre marqueur dans votre scène.

7 – Pour implémenter des évènements personnalisés lors de la reconnaissance du marqueur, vous pouvez étendre la classe DefaultTrackableEventHandler et l'associer à votre objet target.

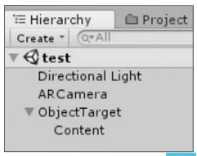
Le résultat lors de la reconnaissance d'une montre avec l'affichage d'une vidéo comme contenu : 7



4



5

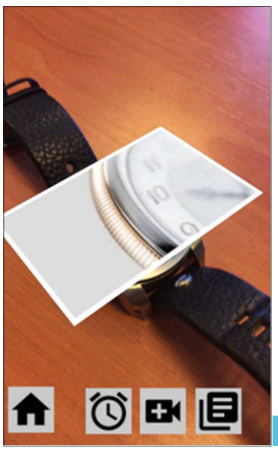


6

Conclusion

Avec cette fonctionnalité de reconnaissance d'objet 3D, Vuforia permet de considérablement augmenter les possibilités de développement d'applications en réalité augmentée. Ce type de reconnaissance est à terme un moyen d'augmenter les possibilités d'interactions autour d'un objet physique et de créer une expérience utilisateur mêlant virtuel et produit de marque. En bref, une belle valeur ajoutée au produit.

Néanmoins, à ce jour Vuforia ne permet pas la détection de surface plane comme ARKit mais depuis l'annonce datant du 2 octobre 2017 annonçant la prise en charge de la détection de surfaces plane, Vuforia sera sûrement en 2018 LA solution AR multiplateforme à suivre. •



7

Sylvain Saurel
sylvain.saurel@gmail.com
Développeur Java / Android
https://www.ssaurel.com

Java 10 : une version enfin à l'heure !

Lancée le 21 septembre 2017, la version 9 de Java aura connu plus de 2 ans de retard par rapport à son planning de lancement initial. Afin d'éviter ces retards récurrents pour les nouvelles versions de Java, Oracle a décidé d'accélérer drastiquement son cycle de mise à jour. Ainsi, une nouvelle version du JDK devrait être lancée tous les 6 mois désormais. Ce nouveau rythme "effréné" ne se fait pas sans certaines concessions comme nous allons le voir.

Dans un monde informatique évoluant de plus en plus rapidement, la plateforme Java est souvent la risée des développeurs utilisant d'autres langages du fait de la lenteur de ses évolutions. Il aura ainsi fallu attendre 5 ans après la sortie de Java 6 en 2006 pour que Java 7 sorte en 2011. Cette dernière version arrivant finalement amputée de nouveautés majeures telles que les lambdas expressions ou Jigsaw, le projet de modularisation du JDK. Trois années supplémentaires furent encore nécessaires pour doter la plateforme de ces fameuses fonctions anonymes ouvrant la voie à la programmation fonctionnelle en Java. Enfin, la modularisation du JDK n'est finalement arrivée que le 21 septembre 2017 avec Java 9, une mouture souvent nommée désir par ses détracteurs.

Ce résumé rapide de l'évolution de Java au cours des dix dernières années montre clairement pourquoi Oracle a décidé de revoir sa stratégie de mise à jour de Java. Désormais, la société de Larry Ellison s'attachera à lancer une nouvelle version du JDK tous les 6 mois. Devant un objectif si ambitieux, il est justifié de se demander comment Oracle pourrait tenir ce délai compte tenu des retards rencontrés par les précédentes versions de Java. Pour que cet objectif soit tenu, Oracle a décidé de réduire le nombre de fonctionnalités et d'améliorations embarquées dans chaque nouvelle mouture.

Le temps où la sortie d'un nouveau JDK s'accompagnait d'une longue liste de nouveautés est donc révolu. Cette nouvelle direction prise par les équipes en charge de Java semble être une bonne décision puisqu'elle garantira un meilleur rythme de mises à jour. Il faudra cependant voir si les entreprises arriveront à tenir cette cadence alors que certaines peinent encore à passer notamment à Java 8.

Introduction des variables locales

Java 10 est donc la première version du JDK adoptant la nouvelle philosophie de mise à jour accélérée. De fait, les fonctionnalités nouvelles sont limitées et auront moins d'impact que celles apportées par Java 8. Du point de vue du développeur, l'introduction des variables locales à inférence de type constitue la nouveauté la plus intéressante et la plus attendue. Elle rajoute le mot-clé var au langage Java et rend possibles les écritures suivantes :

```
var i = 100;
var j = 200;
var k = myClass.getSum(i, j);
var list = new ArrayList<String>();
list.add("Sylvain");
list.add("Saurel");
var stream = list.stream();
```

Dans notre exemple, le type des variables est inféré. Ainsi, la variable *list* est bien de type *ArrayList<String>* et la variable *stream* est bien de type *Stream<String>*. Le code produit par les développeurs Java sera donc allégé avec cette nouveauté.

Néanmoins, le nouveau mot-clé var reste limité aux cas suivants :

- variables locales avec initialiseurs,
- variables servant d'index dans des boucles for,
- scope local d'une boucle conditionnelle.

L'exemple suivant au sein d'une boucle for sera donc valide :

```
for(var i = 0;i < 10;++i) {
  System.out.println("i = " + i);
}
```

A contrario, le nouveau mot-clé ne pourra donc pas être employé dans la liste de paramètres d'une méthode ou comme type de retour pour une méthode. Les constructions suivantes produiront donc une erreur de compilation :

```
public int maMethode(var y) {
  return y;
}
public var maMethode2(int y) {
  return y;
}
```

Bien évidemment, les développeurs devront prendre garde à utiliser de manière intelligente ce nouveau mot-clé en réalisant un nommage le plus parlant possible pour leurs variables afin d'éviter de se retrouver avec un code finalement moins verbeux mais totalement illisible. Pour les développeurs respectant de bonnes pratiques dans

le nommage de leurs variables, l'introduction des variables locales à inférence de type s'avérera précieuse.
Prenons l'exemple de code suivant sous Java 9 :

```
UserService userService = services.findUserService();
List<User> users = userService.users();
Map<User, Address> addressesByUser = userService.addressesByUser();
```

Avec Java 10, ce code pourra être abrégé de la manière suivante :

```
var usersService = services.findUsersService();
var users = usersService.users();
var addressesByUser = usersService.addressesByUser();
```

Moins de verbosité tout en gardant la véritable intention derrière le code écrit. Cet exemple nous rappelle une fois de plus qu'un bon choix dans le nommage des variables s'avère suffisant dans la bonne compréhension d'un code.

Améliorations liées au Garbage Collector

Les autres nouveautés apportées par Java 10 ne sont pas à destination directe des développeurs mais visent plutôt à améliorer la plateforme Java. Parmi celles-ci, deux des nouveautés introduites par Java 10 concernent le Garbage Collector. La première, portée par le JEP 304, introduit une nouvelle interface plus claire pour le Garbage Collector améliorant l'isolation du code source des différents garbages collectors proposés.
Actuellement, les fichiers sources Java liés aux garbages collectors sont dispersés, ce qui rend l'implémentation d'un nouveau Garbage Collector problématique. L'introduction d'une meilleure modularité au sein de la JVM pour cette partie simplifiera la tâche des développeurs de nouveaux garbages collectors.
La seconde évolution liée au Garbage Collector est portée par le JEP 307. Conçu pour éviter les opérations de garbage collection complètes, G1 est devenu le Garbage Collector par défaut de la JVM avec Java 9. Néanmoins, il peut tomber en Full GC lorsque des opérations concurrentes sur les collections ne sont pas en mesure d'obtenir assez de mémoire. Afin d'adresser cette problématique, des changements ont été faits afin de rendre le travail de G1 parallélisable avec le JDK 10.

Divers

Les autres nouveautés de Java 10 sont moins marquantes. On peut cependant citer le JEP 312 introduisant un nouveau mode d'exécution pour les callbacks liés aux threads. Cette nouveauté rend possible l'arrêt individuel de threads alors que jusqu'à présent cela était impossible au niveau interne dans la JVM.
L'allocation mémoire de la Heap sur des périphériques mémoires alternatifs laissés au choix de l'utilisateur est désormais supportée grâce au JEP 316. Ceci s'avérant intéressant pour assigner des processus de faible priorité à la mémoire NV-DIMM alors que les processus de plus grande importance pourront s'appuyer sur la DRAM. Une possibilité fort utile en environnement multi-JVM mais s'adressant avant tout aux spécialistes en optimisation de machines virtuelles Java.
Enfin, le JEP 319 rend open source les certificats root des outils Oracle autour de Java SE. C'est une excellente nouvelle qui facilitera la création de builds OpenJDK tout en réduisant les différences avec les builds de Java réalisées par Oracle. Ainsi, les composants de sécurité tels que TLS (Transport Level Security) fonctionneront par défaut au sein des builds OpenJDK à l'avenir.

Java 11 prévu pour Septembre 2018

Java 10 marque le démarrage du nouveau cycle de livraison des versions du JDK avec une version majeure planifiée tous les 6 mois désormais. De fait, Java 11 est déjà attendu pour Septembre 2018 et nous connaissons d'ores et déjà les fonctionnalités qui seront embarquées. Au total, 4 JEPs ont été retenus. Ce nombre peut sembler faible mais il fait partie intégrante de la nouvelle stratégie d'Oracle pour garantir une sortie en temps et en heure des nouvelles versions du JDK.
Parmi les nouveautés attendues, on notera donc l'extension du scope des variables locales à inférence de type à la liste des paramètres des lambdas expressions via le JEP 323. Intitulé "Dynamic Class-File Constants", le JEP 309 se donne pour objectif de réduire le coût et les perturbations liés à la création de nouvelles formes de constantes de fichiers de classes matérialisables. A la lecture du but de ce JEP, vous aurez bien entendu compris qu'il s'adresse avant tout aux créateurs de nouveaux langages tournant sur la JVM et non à la majorité des développeurs Java.
De nouvelles améliorations au Garbage Collector de la JVM seront apportées par le JEP 318 afin d'améliorer ses performances et celles des programmes Java. Discuté depuis de nombreux mois, le retrait des modules Java EE et CORBA de Java SE devrait enfin être effectif avec Java 11.

Conclusion

La sortie de Java 10 marquera un tournant dans l'histoire de Java avant tout pour le nouveau cycle de mise à jour adopté par Oracle. Dans une industrie informatique évoluant de manière toujours plus rapide, la livraison d'une nouvelle version majeure de Java tous les 6 mois devrait permettre de garantir la pérennité de la plateforme à long terme en la gardant compétitive face à la concurrence. Néanmoins, cette évolution n'est pas sans compromis puisque désormais les nouvelles fonctionnalités apportées par chaque version seront de taille plus petite. En outre, les entreprises devront tenter de suivre ce nouveau rythme de mises à jour. Pas évident, mais c'est à ce prix que la plateforme Java pourra conserver sa position dominante dans l'industrie informatique. Le jeu en vaut donc largement la chandelle. •

Stanislas Dolcini
Software Engineering, Avanade
stanislas.dolcini@avanade.com

avanade

Les 10 erreurs les plus communes en JavaScript

Le langage de programmation JavaScript devient de plus en plus commun de nos jours. Cette popularité grandissante s'explique par l'explosion du web ces dernières années. De nombreux moteurs de jeu web utilisent ce langage, comme Babylon.js, Three.js, et Blend4Web. Unity 3D aussi le supporte aux côtés du C#. On peut aussi citer 0 A.D. un jeu vidéo open source dont le moteur, Pyrogenesis, utilise le JavaScript comme langage de script. La plateforme Node.js utilise aussi ce langage afin de permettre la création d'applications web très performantes.

Ce langage, inventé en 1995 par Brendan Eich en seulement dix jours, a subi de nombreuses améliorations au fil des années. Sa simplicité et sa permissivité sont à la fois ses plus grandes forces et ses plus grandes faiblesses. Voyons ensemble quelques erreurs communes en JavaScript.
Dans l'article suivant :

```
// -> Désigne le résultat d'une expression.
// > Désigne le résultat d'un console.log.
// est un commentaire simple.
```

Erreurs de portée

Saviez-vous que vous pouviez faire quelque chose de ce genre ?

```
var tableau = [0, 1, 2 , 3, 4];
for (var i = 0 ; i != tableau.length; ++i){
  console.writeline(i)
}
console.writeline(i); // > Affiche 5
```

Solution : Si vous utilisez ES6 ou plus récemment ES2015, remplacez quand vous le pouvez 'var' par 'let'. Le mot clef 'let' a l'avantage de ne pas être déclaré en dehors de la portée du block dans lequel il est déclaré.

Oublis de déclaration

Si on reprend l'exemple précédent, le code suivant produira la même sortie :

```
tableau = [0, 1, 2 , 3, 4];
for (i = 0 ; i != tableau.length; ++i){
  console.writeline(i)
}
```

De même pour celui-ci :

```
var tableau = [0, 1, 2 , 3, 4];
for (i = 0 ; i != tableau.length; ++i){
  console.writeline(i)
}
```

Et toutes les variations imaginables.

Solution : Utiliser « "use strict"; » en haut de chaque fichier JavaScript. En tapant le code ci-dessous, vous obtiendrez une erreur lors de l'exécution :

```
Uncaught ReferenceError: i is not defined
```

Re-déclaration de variables

En JavaScript, il est tout à fait possible d'écrire :

```
var lama = 0 ;
var lama = 2 ;
var lama = 3 ;
```

Le code ci-dessus ne génèrera aucune erreur. Dans un fichier assez grand cela peut poser des soucis notamment au niveau de la portée de var. Le code :

```
let lama = 0 ;
let lama = 2 ;
let lama = 3 ;
```

Va générer une erreur :

```
Uncaught SyntaxError: Identifier 'lama' has already been declared at line1:1
```

Solution : Utiliser la nouvelle variable d'ES6 et ES2015 let. Hormis pour de vieilles versions d'Internet Explorer, elle est relativement bien supportée.

Confusions sur l'égalité

Toutes les lignes ci-dessous sont vraies.

```
console.log(false == '0'); // -> true
console.log(null == undefined); // > true
console.log(" \t\r\n" == 0); // > true
console.log('' == 0); // > true
```

Cela vient de la propension de JavaScript à convertir les types, ainsi false devient 0 quand il est transformé en int et de même '0' est transformé en 0.
En revanche, null et undefined sont différents. Convertis en int, ils donnent 0 et NaN (Not a Number) respectivement.

```
console.log(null === undefined); // > false
if ({}) {} // -> true
if ([]) {} // -> true
```

Dans le cas ci-dessus, c'est un peu plus compliqué. Un objet ou un tableau vide, quand ils sont déclarés, retournent true lorsqu'on les entoure d'un block conditionnel.
Cela vient d'une notion spécifique en JavaScript, la notion de truthy et falsy. Un objet défini est truthy. Concrètement, une expression truthy deviendra true dans une instruction if et inversement pour falsy qui deviendra false :

Abonnez-vous à Programmez! Abonnez-vous à Programmez! Abonne

Nos classiques

1 an ..... 49€*
11 numéros

2 ans ..... 79€*
22 numéros

Etudiant ..... 39€*
1 an - 11 numéros

** Tarifs France métropolitaine*

[Programmez!] Le magazine des développeurs

Abonnement numérique

PDF ..... 35€
1 an - 11 numéros

Souscription uniquement sur
www.programmez.com

Option : accès aux archives 10€

Offres printemps 2018

1 an ..... 59€
11 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robuste
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

2 ans ..... 89€
22 numéros
+ 1 vidéo ENI au choix :

- Symfony 3*
  Développer des applications web robuste
  (valeur : 29,99 €)
- Raspberry Pi*
  Apprenez à réaliser et piloter une lumière d'ambiance
  (valeur : 29,99 €)

** Offre limitée à la France métropolitaine*

Toutes nos offres sur www.programmez.com

Oui, je m'abonne

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

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

☐ **Abonnement 1 an :** 59 €
11 numéros + 1 vidéo ENI au choix :
☐ **Abonnement 2 ans :** 89 €
22 numéros + 1 vidéo ENI au choix :

☐ Vidéo : Symfony 3
☐ Vidéo : Raspberry Pi

PROG 218
Offre limitée, valable jusqu'au 1er juin 2018

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

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

E-mail : @

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

*** Tarifs France métropolitaine**

Offres 20e anniversaire !*

1 an - 11 numéros	2 ans - 22 numéros
79,99 €	99,99 €
(au lieu de 147,99 €)	(au lieu de 177,99 €)

+

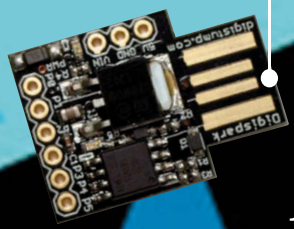
1 clé USB contenant
tous les numéros depuis le n°100

+

4 numéros vintages
(selon les stocks)

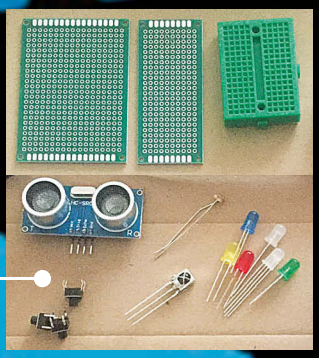
+

1 carte maker Digispark ATtiny84
compatible Arduino



+

1 lot de composants / capteurs
(selon arrivage du jour)



+

1 an de Pharaon Magazine soit 4 numéros :
pour découvrir l'Egypte des Pharaons !

** Offre réservée à la France métropolitaine*

Sur notre site web uniquement : **https://www.programmez.com/catalog/20eme-anniversaire**

```
if ({}) {} // -> truthy donc vraie
if ({} == true) {} // false
if (null) {} // -> falsy donc fausse
if (null == false) {} // -> false
```

Dans le cas de NaN (Not a number), toujours utiliser isNan(valeur) pour éviter des résultats déplaisants.

```
console.log(NaN == NaN);   // > false
console.log(NaN === NaN);  // > false
console.log(isNaN(NaN));   // > true
```

Utilisation incorrecte de « this »

```
Jeu.prototype.redemarrer = function () {
  this.clearLocalStorage();
  this.timer = setTimeout(function() {
   this.clearBoard();
  }, 0);
};
```

Dans ce cas, quelle est la valeur de this dans la fonction setTimeout ? Le code ci-dessous retournera :

```
Uncaught TypeError: undefined is not a function
```

Dans la fonction redémarrer, this est en fait window car l'appel de setTimeout est celui de window.setTimeout.

Solution : utiliser self, pour indiquer à la fonction quelle valeur de this elle doit utiliser.

```
Jeu.prototype.redemarrer = function () {
  this.clearLocalStorage();
  let self = this;
  this.timer = setTimeout(function() {
   self.clearBoard();
  }, 0);
};
```

Erreurs communes de frappe

```
var x = 0;
if (x = 10)
```

Dans ce cas, la condition retournera true, en plus d'assigner 10 à x, parce que 10 == true
On peut aussi citer les soucis avec les guillemets.

```
let value = 'ceci n'est' + 'qu' un' +  'example';
```

Solution : La coloration syntaxique est plutôt utile dans ce cas.

Subtilités des conditions de type « switch »

En JavaScript l'instruction switch effectue une comparaison stricte (l'équivalent de « === »). La comparaison stricte vérifie en plus que les types sont identiques. Dans le cas suivant, '10' et "10" sont différents. L'un est de type char, et l'autre de type string.

```
let chaine = '10';
switch(chaine)
{
  case 10:
    // N'affiche pas de popup avec Hello comme message
    // car la condition n'est pas satisfaite
    alert("Hello");
    break;
  default:
}
```

Dans ce cas, l'instruction switch n'affichera pas de pop-up.

Solution : la fonction parseInt(valeur) ou le symbole « + »

```
let chaine = '10'
switch(parseInt(chaine))
{
  case 10:
    alert("Hello"); // Affiche un popup avec Hello comme message
    break;
  default:
}
```

ou :

```
let chaine = '10'
switch(+ chaine)
{
  case 10:
    alert("Hello"); // Affiche un popup avec Hello comme message
    break;
  default:
}
```

Utilisation de JavaScript sur des balises HTML avant de les afficher

```
<html>
<head>
 <script>
  document.getElementById("myDiv").innerHTML = "This div is my div";
 </script>
</head>
<body>
 <div id="myDiv">This div is your div.</div>
</body>
</html>
```

Ce code va générer une erreur car la div n'existera pas lorsque le code sera appelé.

Solution : appeler la fonction lorsque que la « div » se charge à l'aide du mot clef « onload. »

```
<html>
<head>
 <script>
  function nameMyDiv() {
   document.getElementById("myDiv").innerHTML = "This div is my div";
  }
 </script>
</head>
```

```
<body onload="nameMyDiv();">
 <div id="myDiv">This div is your div</div>
</body>
</html>
```

On notera aussi que déclarer ses fonctions JavaScript dans le code HTML n'est pas recommandé. Il vaudrait mieux dans ce cas avoir un fichier JavaScript à part.

Redéfinition de fonctions

Contrairement à d'autres langages comme le C#, le polymorphisme n'est pas supporté en JavaScript. Il n'est donc pas possible de redéfinir plusieurs fois la même fonction avec des paramètres différents, chaque redéfinition de fonction en JavaScript remplace toutes les occurrences précédentes. Il faut donc faire très attention à ne pas nommer des fonctions comme celles de base en JavaScript, au risque de casser tout le code précédent.
Exemple simplifié :

```
function functionName(){
   console.log("hello"); // > Affiche hello
}
function functionName(value){
   console.log("goodbye"); // > Affiche goodbye
}

functionName(); // > Affiche goodbye
```

On notera qu'on peut appeler la fonction sans son paramètre. Dans ce cas, il serait préférable d'utiliser les paramètres optionnels.
On notera aussi qu'on utilise la syntaxe :

```
function functionName(){
   console.log("hello"); // > Affiche hello
}
```

Et non pas :

```
var myFunction = function(arg1, arg2){
   // > do something here
};
```

La différence est encore une fois la portée de la variable. Une fonction déclarée de la première façon est globale dans le fichier, alors que l'autre ne peut être appelée qu'après avoir été déclarée.

Oubli de points-virgules

L'oubli de point-virgule est de-facto néfaste à votre code. Mais il peut aussi créer des erreurs plus dures à trouver à cause de la façon dont le code est interprété. Dans le cas suivant, à première vue tout se passe bien, pourtant cette fonction retournera toujours undefined.

```
function test(){
   var name = "Hello";
   return // le parseur va ajouter un point-virgule ici
   {
    "Name": name
   }
 }
```

Autre exemple plus compliqué :

```
function test2(){
  var someList = undefined

  console.log("Hi!") // le parseur ne va pas ajouter de point-virgule ici
  (someList || []).map((item) => item)
}
```

Dans ce cas, l'interpréteur JavaScript va tenter de passer la ligne du dessous comme paramètre à la fonction console.log et échouer en retournant.

```
TypeError: console.log(...) is not a function
```

Solution : Considérez le point-virgule comme obligatoire, utilisez des Linters, et ouvrez vos accolades sur la même ligne que vos instructions return ;

Conclusion

De manière générale, il vaut mieux coder en JavaScript comme si le langage n'était pas si permissif. L'utilisation de let est recommandée, ainsi que de « "use strict" ; ». Les linters comme JSHINT, ES Lint peuvent aussi aider à la manière d'IntelliSense sur Visual studio. Il existe aussi des versions en ligne qui permettent de copier-coller le code. On peut mentionner les nombreux éditeurs de texte disponibles sur le marché tels que Visual Studio Code de Microsoft, Webstorm de Jetbrains, Sublime Text par Sublime HQ Pty Ltd, Brackets d'Adobe, Atom de GitHub, Notepad++ de Don Ho, sans oublier Vim et Emacs.
Tous ces précieux outils supportent le linting que ce soit nativement ou par l'intermédiaire de plugins. Ils supportent aussi la coloration syntaxique, et l'autocomplétion. Désormais des plateformes comme Sonar Qube ou encore Jenkins permettent de facilement suivre la qualité du code et le respect des bonnes pratiques. •

Références

PETERSON, R. (2016). ***Buggy JavaScript Code: The 10 Most Common Mistakes JavaScript Developers Make***. [online] Toptal Engineering Blog. Available at: https://www.toptal.com/javascript/10-most-common-javascript-mistakes [Accessed 6 Mar. 2018].
W3schools.com. (2018). ***JavaScript Mistakes***. [online] Available at: https://www.w3schools.com/js/js_mistakes.asp [Accessed 6 Mar. 2018].
MINNICK, C. and Holland, E. (2018). ***10 Common JavaScript Bugs and How to Avoid Them - dummies***. [online] dummies. Available at: http://www.dummies.com/web-design-development/javascript/10-common-javascript-bugs-and-how-to-avoid-them/ [Accessed 6 Mar. 2018].
WEYL, E. (2010). ***16 Common JavaScript Gotchas***. [online] Standardista. Available at: http://www.standardista.com/javascript/15-common-javascript-gotchas/#overload [Accessed 6 Mar. 2018].
KARATZAS, T. (2017). Avoid These Common JavaScript Mistakes – codeburst. [online] codeburst. Available at: https://codeburst.io/avoid-these-common-javascript-mistakes-ec7fbb642e8b [Accessed 6 Mar. 2018].

Alexis "Horgix" Chotard
est consultant chez **Xebia** (**https://xebia.fr**), *principalement autour des sujets DevOps et Cloud Native.*

Infrastructure-as-code : pourquoi et comment coder son infrastructure ?

Historiquement, les serveurs peuvent être répartis en deux catégories : **serveurs physiques et machines virtuelles**. Tous deux étaient installés manuellement, en démarrant sur une image d'installation. Il était ensuite possible de créer des *snapshots*, images instantanées de disques de machines virtuelles existantes, afin de les utiliser comme base pour ne pas réinstaller encore et toujours la même chose. Cependant ces images, ou *templates*, montrent vite leurs limites. Dès qu'il est nécessaire d'y apporter une modification, celle-ci se fait manuellement... posant des problèmes en termes de traçabilité, de répétition (recréation du template en cas de perte) ou tout simplement de connaissance : il est nécessaire de documenter ce qui est pré-installé sur ces images, de tenir la documentation à jour, et autres tâches fastidieuses et sources d'erreur.

Et si nous décidions de **raisonner dans le sens inverse** ? Écrire la documentation, puis avoir **des outils en mesure de la comprendre pour configurer nos serveurs**, générer nos images, et autres tâches qui n'ont aucune valeur ajoutée à être faites manuellement ? Parlons donc d'infrastructure-as-code.

Note

Dans le reste de cet article, nous porterons notre attention sur git, celui-ci étant aujourd'hui le plus outillé des gestionnaires de versions, ce qui sera un axe clé de notre infra-as-code.

Qu'est-ce que l'infrastructure-as-code ?

Que représente de l'infrastructure "comme du code" ?

Commençons tout d'abord par voir **ce qu'est du code aujourd'hui** :

- Le code n'est, au final, qu'un **descriptif textuel d'instructions** dans un langage donné.
- Cette description textuelle est **comprise et transformée en actions concrètes**, soit par un interpréteur soit via un compilateur.

Cette description textuelle peut (et doit) être versionnée : que ce soit via git, SVN, Mercurial ou autre.

Ce *versionnement* apporte un certain nombre de bénéfices. En effet, dans bien des cas, il existe toujours un point centralisant la base de code et permettant aux développeurs de regrouper et synchroniser leurs contributions au projet, typiquement Github ou Gitlab pour un projet versionné via git (qui est pourtant un système de versionnement distribué). Ce point central devient alors le **point névralgique des actions découlant de la contribution de code** : build, test, voire déploiement du code. Cette centralisation apporte également une certaine **traçabilité** : en effet, on possède un historique des modifications, qui sont datées et associées à un auteur, voire potentiellement signées cryptographiquement. Par ailleurs, on s'efforcera de ne jamais modifier l'histoire passée une fois celle-ci centralisée (git rebase puis git push --force par exemple), assurant ainsi l'**intégrité** de cet historique. Par ailleurs, "la doc, c'est le code" est un credo qui revient de plus en plus fréquemment, notamment via le mouvement Craftsmanship.

L'infrastructure-as-code va donc consister à décrire l'infrastructure voulue textuellement, comme du code, et à versionner ce code, bénéficiant ainsi d'avantages notables tels qu'un point central pour l'automatisation de la gestion de notre infrastructure, la traçabilité de l'historique des évolutions, l'approbation de modifications ou encore une certaine forme de documentation.

En revanche, une fois cette description textuelle écrite, qu'en fait-on ? Revenons sur notre comparaison avec le code : une fois écrit, du code est soit interprété, soit compilé. Il y a donc des **outils** en mesure de **comprendre cette description textuelle d'instructions** pour en tirer des actions / exécutions concrètes. Dans le cas d'un langage interprété tel que Python il s'agira de l'interpréteur Python lui-même, et dans le cas de langages compilés tels que le C il s'agira d'un compilateur tel que GCC ou Clang.

Pour notre infrastructure, c'est la même chose : une fois cette description textuelle écrite, il est nécessaire de posséder des outils en mesure de comprendre cette description pour les transformer en actions concrètes, sans quoi il ne s'agirait que de simple documentation.

Nous allons donc voir dans cet article quels sont ces outils, les paradigmes associés, les grands mouvements de l'infrastructure-as-code, ou encore le résultat que l'on peut aujourd'hui, en 2018, espérer atteindre avec de telles pratiques.

Pour le reste de cet article, nous vous simplifions la lecture en nous contentant de "*infra-as-code*"

Un peu de théorie : l'idempotence

En mathématiques et en informatique, l'idempotence signifie qu'**une opération a le même effet qu'on l'applique une ou plusieurs fois**. Par exemple, la valeur absolue est idempotente abs(abs(-5)) = abs(-5) = 5.

Dans notre cas, ce sont les actions de notre outil d'infra-as-code que nous voulons aussi idempotentes que possible : nous préférons donc par exemple "nous assurer de la présence d'une ligne dans un fichier / nous assurer de l'existence d'une machine virtuelle" à "ajouter une ligne dans un fichier / ajouter une machine virtuelle" ; l'idée étant de pouvoir lancer notre outil en boucle sans que l'état ne change en l'absence de modifications dans le code décrivant l'infrastructure.

Cette notion sera cruciale pour comprendre certains choix et certaines limitations des solutions qui vont vous être présentées.

En somme : **on ne décrit pas des actions ; on décrit un état souhaité, puis notre outil va inférer les actions à effectuer afin de converger vers cet état**.

Infrastructure vs Configuration

L'*infra-as-code* englobe deux types de descriptions :

- La description de l'**infrastructure** hébergeant une application : machines virtuelles, configuration des composants réseaux, etc.
- La description des **services** déployés et hébergés sur cette infrastructure sous-jacente (ou environnement logiciel) : paquets installés, fichiers de configuration, bibliothèques dont l'application dépend, etc.

Ces deux pans de l'infra-as-code travaillent avec des entités logiques différentes (machines, réseau, routeur *versus* bibliothèques, paquets, applications) et demandent une gestion de l'idempotence différente, ce qui implique l'utilisation d'outils différents pour chacune d'entre elles.

Commençons par des exemples de description d'infrastructure, ici avec ***Terraform***, outil permettant de décrire l'organisation des machines sur plusieurs fournisseurs de cloud tels AWS (Amazon Web Services) :

```
resource "aws_instance" "webserver" {

  # Le type (dimension) d'instance à utiliser
  instance_type = "t2.micro"

  # L'image disque à utiliser pour démarrer la machine
  ami = "ami-5431cf76"

  # Le nom de la paire de clés SSH à déployer sur la machine
  key_name = "xebia-key-pair"
}
```

Dans cet exemple, nous décrivons une instance EC2 chez AWS de type *t2.micro*, sur laquelle nous pourrons nous connecter en SSH en utilisant la paire de clés SSH xebia-key-pair. Nous pouvons remarquer deux choses : cette description aurait pu être lue sans explication et elle peut être réappliquée pour reproduire la même infrastructure sans aucune différence, ce qui permet par exemple de cloner facilement un environnement de production.

Nous remarquons également qu'il n'y a aucune directive concernant les briques logicielles installées sur cette machine, ce qui est normal : Terraform se contente de décrire l'infrastructure et ses interconnexions. La description de l'environnement sur la machine est déléguée à d'autres outils plus adaptés comme SaltStack ou Ansible.

Voici en comparaison un exemple de description d'un environnement logiciel sur la machine *webserver* décrite précédemment, avec **Ansible**.

```
---
- hosts: webserver
  tasks:
    - name: "Install nginx"
      package:
          name: nginx
          state: installed
    - name: "Ensure nginx is started and enabled"
      service:
          name: nginx
          enabled: yes
          state: started
```

Nous l'aurons compris en lisant ces lignes : nous souhaitons avoir un serveur web nginx sur le serveur, démarré et qui sera automatiquement actif au démarrage. Une nouvelle fois, cette description peut se lire sans avoir de connaissances Ansible.

Nous remarquons bien une séparation logique entre l'infrastructure sous-jacente et la configuration des services qui reposeront dessus, même si la limite peut paraître un peu floue. Depuis l'avènement de l'*infra-as-code*, plusieurs outils ont émergé pour ces deux facettes, et de nouveaux continuent sans cesse d'apparaître.

Les outils décrivant les environnements sont historiquement plus anciens que ceux gérant l'infrastructure car ils ont été (et sont toujours !) appliqués dans un contexte *on-premise*, alors que les autres sont intimement liés au *cloud*. Faisons un rapide tour d'horizon des outils de ces deux grandes familles.

Gestion de configuration

Server/agent vs agent-less

Une distinction importante est à noter parmi les outils de gestion de configuration :

- ceux fonctionnant avec un serveur central ainsi qu'un agent présent sur chaque serveur cible,
- ceux simplement utilisables en ligne de commande, dits sans agent (*agent-less*).

Chaque approche possède son lot d'avantages et d'inconvénients, dont voici les principaux :

Permissions

Les outils de gestion de configuration **sans agent** ont le bénéfice d'être utilisables très rapidement : souvent, ceux-ci n'ont besoin que d'une connexion SSH aux serveurs qu'ils doivent administrer et iront s'y connecter de cette manière pour appliquer les changements nécessaires. Ce point a le bon goût de **déléguer toute la partie gestion de droits à cet accès SSH** : votre outil ne pourra pas effectuer de modifications sur une machine à laquelle vous n'auriez pas accès. De même, une fois connecté sur cette machine en tant que votre propre utilisateur, votre outil utilisera des mécanismes standards d'élévation de privilège tels que sudo lorsque ceci est nécessaire : ainsi, si vous n'avez pas la possibilité de passer *root* sur une machine cible, l'outil ne le pourra pas non plus.

À l'opposé, les outils de gestion de configuration **avec un serveur central** et des agents vont devoir **gérer eux-mêmes la notion de permission** : il s'agit souvent d'une partie complexe, que l'on veut à la fois simple mais suffisamment granulaire. Le côté "granularité" est cependant souvent mis de côté, laissant ainsi quiconque à accès à ce serveur central la possibilité d'y lancer n'importe quelle modification ciblant n'importe quelle machine gérée.

Modifications concurrentes

Second point et non des moindres : dans le cas d'un serveur centralisé, il est possible de garantir qu'une seule source d'infra-as-code est en cours d'application. Mais dans le cas d'outil sans agent, rien ne garantit que deux personnes n'iront pas lancer l'application de configurations différentes simultanément : ainsi, le dernier arrivé écrasera les modifications de son prédécesseur alors que celui-ci pensera que tout a bien été appliqué.

Le versionnement à la rescousse

La réponse à ces problématiques de permission et de concurrence pour les outils de gestion de configuration sera dans la plupart des cas le **versionnement**, plus précisément la centralisation de celui-ci.

Tout comme l'on centralise le code d'une application via Github ou Gitlab, nous allons nous en servir pour le code de notre infrastructure. C'est donc ce dépôt Git centralisé qui sera le point d'entrée de l'application de toute modification : par exemple, lorsque quelqu'un contribue du code à l'infrastructure en le poussant sur la branche *master* du dépôt, un processus d'intégration continue viendra lancer l'outil faisant converger les services vers l'état décrit.

Ordre d'exécution

Une autre différence primordiale réside dans la manière dont les outils de gestion de configuration vont approcher l'état qui leur est décrit. Celui-ci est composé de différents blocs, qui peuvent être **exécutés séquentiellement** (approche la plus simple) ou **ordonnancés intelligemment** (approche la plus optimisée mais aussi la plus complexe).

Dans le premier cas, l'outil se contentera de lire les tâches ou états désirés l'un après l'autre et de faire converger la machine cible vers cet état, étape par étape.

Dans le second cas, l'outil ira lire l'intégralité des états désirés, détecter les dépendances entre certains et tentera ensuite de les paralléliser un maximum. Il en résulte un temps d'exécution souvent drastiquement réduit mais également une complexité de compréhension plus grande.

Maintenant que ces caractéristiques sont posées, voyons quels sont ces outils de gestion de configuration.

Ansible

Ansible (https://ansible.com), écrit par **Michael DeHaan** en **2012**, est désormais maintenu par la communauté et par RedHat qui a racheté la marque en 2015.

Sa principale caractéristique est son approche **impérative et séquentielle**, laquelle permet de décrire une suite d'actions à appliquer sur un groupe de serveurs. 1

L'ordre des tâches ici est explicite, nous devons forcément avoir le paquet *nginx* installé avant d'activer son service système et de le démarrer. Cet aspect séquentiel et exécuté au travers d'une **connexion SSH** le rapproche énormément d'un "simple" framework de scripting, ce qui a grandement facilité son adoption.

Ansible est aujourd'hui l'outil de gestion de configuration as code **le plus populaire**. En effet, sa simplicité, de par son modèle sans agent, la qualité de sa documentation et la variété de ses modules en font un **candidat idéal pour des premiers pas dans l'infra-as-code**. Renforcé par le récent rachat par RedHat, il apparaît comme le leader des gestionnaires de configuration.

On notera également **Ansible Tower**, récemment open-sourcé par RedHat sous le nom d'**AWX** (https://github.com/ansible/awx) et qui vise à fournir une éventuelle approche de serveur centralisé à Ansible, se positionnant au niveau d'orchestrateurs de jobs tels que **Rundeck** (http://rundeck.org/).

SaltStack

SaltStack (https://saltstack.org/), communément abrégé "Salt", a été écrit par **Thomas S. Hatch** en **2011** et est maintenu par la communauté ainsi que la société SaltStack. Il s'agit d'un outil suivant une approche descriptive : l'état à atteindre est décrit et l'ordre d'exécution des opérations est laissé à l'outil. Cette approche paraît plus lisible car elle décrit simplement l'état final et non les actions à faire pour y parvenir 2

L'ordre des tâches est ici implicite, Salt ayant en interne l'intelligence d'effectuer les tâches d'installation (package) avant les tâches de services (service). Salt est le plus souvent utilisé avec un serveur master communiquant avec un agent (minion) sur les machines cibles mais peut tout aussi bien fonctionner en simple ligne de commande comme Ansible. Sa versatilité est sans doute son grand point fort.

Puppet

Puppet (https://puppet.com), de **PuppetLabs**, est le troisième plus connu des outils de gestion de configuration. C'est aussi le plus ancien, sa première version étant sortie en 2005. Il suit la même approche que Salt mais fournit un *Domain Specific Language* (DSL) pour aider à cette tâche. 3

Cette fois, nous devons expliciter les dépendances, Puppet permettant de créer des relations de dépendances via les mots clés *require* et *notify*. Dans l'exemple, le service *nginx* sera démarré après l'installation de *nginx* et toute mise à jour du paquet *nginx* engendrera un redémarrage du service *nginx*.

Les autres

Ne pouvant détailler tous les outils dans cet article, notez cependant les noms suivants :

Chef - https://www.chef.io/

Mgmt - https://github.com/purpleidea/mgmt

Capistrano - http://capistranorb.com/

Rudder - https://www.rudder-project.org/

```
- name: "Install nginx"
  package:
      name: nginx
      state: installed
- name: "Ensure nginx is started and enabled"
  service:
      name: nginx
      enabled: yes
      state: started
```
1

```
webserver:
  pkg.installed:
    name: nginx
  service.running:
    name: nginx
    enable: true
```
2

```
package { 'nginx':
  name => 'nginx',
  ensure => 'installed'
  notify => Service['nginx']
}

service { 'nginx':
    name => 'nginx',
    enable => true,
    ensure => running
    require => Package['nginx']
}
```
3

Comparatif

Outil	Serveur centralisé ou agent-less	Intelligence d'ordonnancement	Facilité de prise en main	Les avantages	Les inconvénients
Ansible	Agent-less ; centralisation possible via Tower/AWX	+	+++++	Documentation Facilité de prise en main	Utilisation à tort et à travers Cas d'usages limités
SaltStack	Les deux	++++	++	Puissance d'expressivité Performances	Complexité
Puppet	Serveur	++	+++	Répandu	Performance

Gestion d'infrastructure

La gestion d'infrastructure automatisée et gérée comme du code a vu son avènement avec celui du cloud. En effet, redéfinissons brièvement ce qu'est le **cloud** : il s'agit de **l'abstraction de ressources et de leur mise à disposition via des APIs**. Que l'on parle de cloud public ou de cloud privé, cette définition s'applique. Et c'est justement ces APIs qui vont nous permettre de gérer notre infrastructure comme du code : les actions concrètes découlant de notre description d'état souhaité ne seront au final que des **appels aux APIs de notre fournisseur de Cloud**.

Terraform

Terraform (https://www.terraform.io/) fait partie des outils créés par **Mitchell Hashimoto** et qui a donc rapidement rejoint, en **2004**, les outils de la galaxie **Hashicorp**. Voulant couvrir un large spectre de fournisseurs de cloud et y réussissant en partie grâce aux contributions communautaires, il gagne vite en popularité pour être aujourd'hui un des premiers outils cités dès lors que l'on parle d'infra-as-code.

```
provider "google" {
  project     = "xebia-programmez"
  region      = "eu-west2"
}

resource "google_compute_instance" "default" {
  name = "instance"
  machine_type = "n1-standard-1"
  zone = "eu-west2-a"

  boot_disk {
    initialize_params {
      image = "centos-cloud/centos-7"
    }
  }

  network_interface {
    network = "default"
  }
}
```

Comme on peut le voir, la description pour des éléments d'infrastructure est tout aussi claire que celle des configurations évoquées précédemment : on comprend sans complication qu'une instance de taille *n1-standard-1* sera créée dans la zone *eu-west2-a* de GCP (Google Cloud Platform - https://cloud.google.com/) et qu'elle sera basée sur une image pré-existante de CentOS 7. Pour ce faire, Terraform ira directement appeler les APIs de GCP avec nos propres permissions.

Se voulant multi-cloud, Terraform a une forte notion de *provider* (AWS, GCP, Azure ou OpenStack) et pourrait se résumer en l'abstraction d'appels d'APIs vers ces providers via des fichiers déclaratifs. Ceci réduit considérablement la complexité de changement de plateforme de cloud, n'ayant « plus qu'à » trouver la correspondance entre les ressources de chaque fournisseur et à éditer la configuration de Terraform en conséquence.

Seul bémol : Hashicorp utilise son propre langage de description de configuration, le HCL (https://github.com/hashicorp/hcl) et non un format plus standard tel que le JSON ou le YAML.

AWS CloudFormation

CloudFormation est un outil de gestion d'infrastructure par et pour **AWS** et créé en **2011**. À l'opposé de Terraform qui se veut indépendant et multiplateforme, CloudFormation est fourni par AWS et s'y limite.

Là où Terraform est un outil en ligne de commande, CloudFormation est quant à lui un service : une mise à jour de l'infrastructure consiste ici à soumettre au service le descriptif de l'état souhaité, lequel se chargera alors de l'appliquer en appelant les différentes APIs d'AWS pour créer, modifier ou supprimer les ressources en conséquence. Ces actions sur les ressources vont d'ailleurs être effectuées en notre nom et avec nos propres permissions, qui sont déléguées à CloudFormation durant le temps de mise en place de l'état demandé.

Le descriptif ci-dessous sera par exemple envoyé à l'API de CloudFormation :

```
AWSTemplateFormatVersion: '2010-09-09'
Description: "Xebia - Programmez"

Resources:
  MyInstance:
    Type: AWS::EC2::Instance
    Properties:
      AvailabilityZone: eu-west-1a
      ImageId: ami-1b791862
```

CloudFormation ira alors créer une instance via l'API d'EC2, tout comme nous aurions pu le faire nous même via cette dernière. Là encore, le code est explicite et se suffit à lui-même. Notons que la description peut être écrite en JSON ou en YAML. Ici il s'agit de YAML, souvent préféré pour ce genre de description de par sa clarté et la possibilité d'y ajouter des commentaires pour éventuellement expliquer certains choix.

Stockage de l'état

Un point concernant ces outils de gestion d'infrastructure est cependant à noter : ces outils ont besoin de **savoir de quel état ils**

partent afin de pouvoir calculer les opérations à réaliser, une sorte de "diff" entre l'état précédent et l'état demandé.
Du côté de Terraform, **cet état est stocké dans un fichier "*tfstate*"**. Mais qu'advient-il lorsque plusieurs personnes souhaitent travailler sur l'automatisation de la même infrastructure ? Si ce *tfstate* est local, alors chaque personne risquera de se « marcher sur les pieds » et d'appliquer des modifications contradictoires à l'infrastructure.
Il devient donc nécessaire de **synchroniser cet état** entre les différentes personnes, voire d'y ajouter une notion de *lock* afin qu'une seule description ne puisse être appliquée à la fois. Qu'adviendrait-il si une personne indiquait vouloir 5 Load Balancers tandis qu'une autre en demanderait 10 ? On se retrouve dans un cas classique de besoin d'atomicité ; notre transaction souhaitée atomique étant la mise à jour de l'infrastructure avec une définition donnée. Terraform propose une notion de "*remote state*", ou état distant, qui sera par exemple stocké sur AWS S3 avec un *lock dans* DynamoDB. CloudFormation, quant à lui, nous est complètement abstrait et gère tout ceci en interne : il est impossible de lancer 2 mises à jour simultanées sur une même "*stack*" CloudFormation.

Les autres

D'autres solutions sont également à considérer comme pour la partie gestion de configuration, telles que :
Packer - https://packer.io
Cobbler - https://cobbler.github.io
Foreman - https://theforeman.org

Comparatif

Outil	Les avantages	Les inconvénients
Terraform	Multiplateforme	Gestion de l'état
CloudFormation	Gestion du state intégrée Capacité à rollback	Limité à AWS Non Open-Source

Tendances, enjeux et bonnes pratiques

Un peu de DevOps

On confond de plus en plus souvent DevOps et automatisation. Sans rentrer dans un laïus sur tout le mouvement DevOps, celui-ci est souvent résumé par l'acronyme **CALMS** : **C**ulture, **A**utomatisation, **L**ean, **M**esure et Partage (**S**haring). L'automatisation n'en est qu'une des composantes et est en réalité un pilier **facilitant l'adoption** de cette manière de travailler. Pour cause : l'infra-as-code va devenir un axe de collaboration primordial des développeurs et des opérationnels. Quoi de plus simple que d'envoyer une simple "Pull Request" plutôt qu'un ticket ou un e-mail lorsque l'on souhaite modifier quelque chose ? Quelle meilleure manière de collaborer que sur du code, qui sera exécuté tel qu'il est écrit ? L'idée est de livrer du code, que ce soit celui de l'application ou de l'infrastructure, et que l'automatisation fasse le reste !
C'est pourquoi ce mouvement d'infra-as-code vient tout **aussi souvent du côté des développeurs que du côté des administrateurs système** : les premiers y voient une manière de prendre une certaine indépendance tandis que les seconds y trouvent une manière de se simplifier le quotidien de tâches répétitives et souvent à faible valeur ajoutée. Mais rappelons-le : l'objectif de la mouvance DevOps est d'**aligner les équipes sur les objectifs métiers en améliorant la collaboration entre Développeurs et Opérationnels**. Il est donc crucial que ces populations n'automatisent pas chacune "dans leur coin", mais travaillent ensemble à l'élaboration d'une infra-as-code commune comme étant la propriété collective.

Quand votre infrastructure-as-code devient votre documentation

Au final, votre description d'infrastructure, comme son nom l'indique, décrit toute votre infrastructure... un peu comme de la documentation. **Pourquoi tenter de documenter séparément ce qui est décrit par votre code ?** Le seul risque est de voir la documentation se désynchroniser et finir par ne plus être à jour et à l'abandon. La plupart des outils d'infra-as-code ont le bon goût d'utiliser des syntaxes relativement claires et descriptives (c'est d'ailleurs leur but), alors profitez-en ! Ceci ne signifie pas non plus l'éradication de toute documentation : comme pour du code, il peut être pertinent de documenter l'utilisation qui peut en être faite, les choix et compromis qui ont été faits, ou encore l'architecture globale.
Pour aller plus loin, vous pouvez aller jusqu'à **générer votre schémas d'architecture** à partir de votre code, et pour cause : tout y est décrit. Jetez donc un œil à Terraform graph ou encore à l'éditeur de Stack CloudFormation en ligne. Ces outils ayant la connaissance des liens entre les éléments, il leur est parfaitement possible de vous générer un diagramme représentant le tout, bien que pour être honnête, cette génération n'est pas encore suffisamment configurable : nous nous retrouvons rapidement avec des schémas bien plus détaillés que ce que l'on souhaiterait réellement.

Tests

Il y a peu de chances que, dans les tests unitaires de votre code métier, vous re-testiez le fonctionnement des bibliothèques que vous utilisez. Vous ne voulez pas tester *print()* n'est-ce pas ?
Côté infra-as-code, il est aisé de tomber dans ce piège : certains outils tels que Serverspec ou Testinfra proposent de réaliser des tests sur de l'infrastructure. Vous pourriez même être tenté de réaliser votre infra-as-code en **TDD** (https://en.wikipedia.org/wiki/Test-driven_development) : écriture des tests d'infrastructure, puis implémentation de l'infra-as-code et refactoring éventuel. Mais au final, ce serveur en train d'être configuré a forcément une raison d'être : bien souvent, d'y héberger une application. Alors pourquoi ne pas simplement lancer les **tests fonctionnels de l'application** ? C'est le résultat final attendu : une application qui fonctionne.
Les tests sur de l'infra-as-code ne sont donc pas forcément à éliminer totalement, mais il faut en revanche **faire attention à ne pas re-tester ses outils** et s'en contenter pour effectuer de **simples *smoke tests*** éventuellement. N'oubliez pas que ces outils d'infra-as-code ont déjà leur propre batterie de tests automatisés, apportant leur lot de garanties.

Intégration continue

Cette absence de tests ne veut pas non plus dire une confiance aveugle en ces outils. Ceux-ci peuvent comporter des bugs (et en comportent). Afin de s'en prémunir, il est nécessaire, comme pour une application, de **tester les modifications dans un**

environnement hors production. Ainsi, même si les tests automatisés seront *in-fine* ceux de l'application et éventuellement quelques *smoke tests* d'infrastructure, un lancement de ces tests peut très bien n'être motivé que par des modifications d'infrastructure et non de modifications applicatives.
Bien souvent sur du cloud, nous irons jusqu'à recréer quasi-intégralement l'infrastructure. Pour ce qui est des fournisseurs de cloud eux-même, ceux-ci ont évidemment des tests d'infrastructure.

Craftsmanship : l'infrastructure aussi à besoin d'artisans !

Avec cette intégration continue, tout comme pour du code, un certain nombre de bonnes pratiques s'impose. On pensera notamment au mouvement **Craftsmanship**, auquel l'infra-as-code ne fait pas exception. C'est bien souvent le côté qui est mis en défaut côté Opérationnel, n'ayant pas nécessairement tout le recul et la culture de développeurs sur ce sujet. **L'*infra-as-code* implique les mêmes bonnes pratiques que le code** :

- collaboration : peer-review, peer-programming,
- propriété collective du code de l'infrastructure,
- amélioration continue,
- etc.

Pet vs Cattle

Afin de mieux cerner certains patterns de cette infra-as-code, penchons-nous désormais sur l'approche de la gestion des machines, des serveurs eux-mêmes.
Traditionnellement, les serveurs peuvent être comparés à des **animaux de compagnie** (**"pet"** en anglais). Souvent, l'équipe responsable de l'infrastructure interne leur donne même des "petits noms" : dieux de la mythologie grecque, de personnages de bande dessinée ou encore noms d'acteurs. Et pour cause : ces équipes prennent soin de ces serveurs au quotidien comme nous le ferions avec des animaux de compagnie, d'où ces surnoms. Ils les soignent / réparent en cas de problème et connaissent leurs habitudes. Bien souvent, un membre de l'équipe de gestion de cette infrastructure interne est capable de vous annoncer sans sourciller des choses telles que "*Oui c'est un problème habituel, Poséidon est régulièrement surchargé et nécessite d'être redémarré tous les lundis matin*" : exactement comme un animal de compagnie.
Pire encore, dans certaines entreprises on trouve encore une absence de noms associés à des serveurs, desquels les administrateurs connaissent directement les adresses IP. La tendance à l'évolution vers une vision de masse devient évidente dès que l'on adresse ce point en particulier : il est peut-être possible de "mémoriser" voire de dicter des adresses IPv4, telles *172.42.13.37*, mais ceci devient d'office impossible lors l'utilisation d'adresses IPv6 telles *2001:0db8:0000:0042:0000:8a2e:0370:7334*, qui sont justement la conséquence d'un nombre de terminaux / serveurs en pleine expansion.
L'approche moderne se veut davantage orientée vers une gestion type **bétail** (**"cattle"** en anglais). L'objectif est de gérer l'infrastructure telle un troupeau : vos serveurs se voient simplement assignés un numéro, et vous les gérez comme un tout. Il n'est pas triste d'en voir disparaître, et si l'un tombe "malade" vous le remplacez simplement. Ce "remplacement" dans un contexte plus concret se traduirait par un pur remplacement d'une machine virtuelle ou par le re-provisionnement du serveur physique concerné.
Partez de cette supposition : chaque serveur doit pouvoir se voir assigner un identifiant aléatoire et unique. Ceci n'exclut cependant pas d'avoir certaines différenciations ! Éventuellement, ces serveurs se verront assigner des rôles ou des étiquettes en fonction de leurs capacités, par exemple la présence de GPU.

Multiplateforme

Une question revient fréquemment : **mes outils sont-ils multiplateformes ?**
Pour les outils de gestion d'infrastructure, la question est plus précisément : suis-je **agnostique de la plateforme de Cloud ciblée ?** Dans le cas de CloudFormation, celui-ci étant dédié à AWS, la réponse est bien évidemment non. Pour ce qui est de Terraform, c'est un peu plus ambigu ; Terraform sera capable de gérer de l'infrastructure de différents fournisseurs comme expliqué précédemment, mais il ne sera pas possible de prendre une configuration faite pour AWS et de l'appliquer sur GCP sans modification. Les services ont beau être en de nombreux points similaires, ils ne le sont pas en réalité. Même si Terraform pouvait nous abstraire la différence entre les APIs, une définition de Load Balancer n'aura pas les mêmes paramètres d'un fournisseur à l'autre. Il n'est donc pas possible d'être multiplateforme "comme par magie" avec Terraform. En revanche, celui-ci permet d'avoir un langage commun pour décrire ces infrastructures sur de multiples plateformes, ce qui en facilite grandement l'administration.
Pour ce qui est de la gestion de configuration, ce questionnement s'intéressera au **système d'exploitation ciblé**. Par exemple, entre différentes distributions Linux, le gestionnaire de paquets ne sera pas le même, les services seront gérés par sysvinit ou systemd, les chemins de configuration par défaut ne seront pas les mêmes, etc. Mais les outils de gestion de configuration abstraient souvent très bien ces différences, et rendent possible l'exécution conditionnelle en fonction du système ciblé, nous permettant ainsi de gérer nous-mêmes les différences entre les plateformes cibles.

Sécurité / sûreté

Une question revient pour toute technologie ou paradigme : **est-ce un problème de sûreté ?** La réponse dans le cas de l'infra-as-code est claire : **Non, au contraire !** Vous gérez votre infrastructure comme du code d'un point de vue sûreté : les avantages de Git ne sont plus à citer ; votre dépôt Git est votre source de vérité. Chaque commit peut être signé et soumis à approbation avant d'être intégré au système pour que ces modifications soient appliquées automatiquement.
Le seul point où gérer son infrastructure comme du code pourrait représenter un risque de sécurité est lorsque les outils utilisés pour cette gestion sont bugués : c'est pourquoi il est malgré tout nécessaire, comme pour du développement, de tester ses modifications sur un environnement hors production, pour ensuite, comme pour une application, les appliquer en production. Un processus automatisé restera toujours moins risqué que des actions manuelles.
Pour l'intégration des changements à la base de code de votre

infrastructure, on se reposera sur la notion d'authentification et d'autorisation du serveur centralisant le dépôt Git contenant la description.
Pour ce qui est de l'application d'état, on trouvera par exemple dans Kubernetes toute une gestion d'autorisation basée sur des rôles (RBAC - https://kubernetes.io/docs/admin/authorization/rbac/).
Au final, *tout* est programmatique, il devient donc impossible de contourner un quelconque processus d'autorisation, là où de manière traditionnelle (par demandes orales, tickets, e-mails ou autres) il devient très rapidement possible de laisser une partie du processus de côté.

Conteneurisation

C'est désormais un fait, la conteneurisation gagne du terrain et s'impose de plus en plus, jusqu'en production. Nous prendrons ici l'exemple le plus populaire : **Docker**.
L'avantage de **Docker** n'est pas tant l'outil en lui-même, mais l'écosystème qui vient avec. En effet, **tout est pensé dans une optique d'infra-as-code** depuis la base !

Infrastructure immuable

C'est le moment pour une petite parenthèse sur un concept qui devient de plus en plus crucial : celui d'infrastructure immuable (ou "*imutable*" en anglais). Le constat est le suivant : changer des configurations au *runtime* d'une machine est assez incertain : on part d'un état **potentiellement inconnu**, et on se retrouve rapidement avec un empilage de couches de modifications successives.
Les machines virtuelles et l'infra-as-code nous permettent aujourd'hui de facilement re-créer des images de systèmes entiers. Alors **pourquoi cette image d'un système tout entier ne deviendrait pas notre livrable ?** Ainsi, au lieu de se contenter de livrer le code et ses dépendances, on livrerait le code, ses dépendances, l'OS qui va avec et tous les composants nécessaires afin de n'avoir qu'à démarrer une nouvelle instance de cette image pour obtenir une application fonctionnelle.
On pourrait construire de telles images manuellement sur de la virtualisation classique :

1- Créer une instance temporaire à partir d'une base standard.
2- S'y connecter et y configurer tout ce qui est nécessaire à l'application : binaire ou code de mon application, dépendances, paquets, fichiers de configuration, ...
3- Arrêter cette instance temporaire et la transformer en image, qui n'est autre qu'une sorte de *snapshot*.

Mais nous ne sommes pas ici pour réaliser ces étapes manuellement : des outils tels que **Packer** et **Ansible** nous permettent de décrire ces étapes comme du code. Packer s'occupe des étapes 1 et 3, et appelle Ansible lors de l'étape 2. Souvenez-vous : avoir des outils qui ne font qu'une chose mais la font bien !
Cette image, qui est au final **l'artefact résultant du build de mon application** sera ensuite déployée sans modification au *runtime*. On a ainsi la garantie que ce qui est testé est exactement ce qui sera déployé en production, et on élimine ainsi un nombre certain de risques.
La notion d'images de conteneurs n'est en réalité qu'une version plus évoluée et légère de cette infrastructure immuable : **une image Docker n'est au final qu'un *rootfs*** qui servira de base au processus conteneurisé. Lors d'une nouvelle version de l'application, on reconstruira l'image y correspondant, puis cette image sera livrée et déployée.

Construction d'images

Une image Docker peut être construite via un fichier décrivant textuellement de quelle base partir et quelles actions appliquer. Si vous ne tiltez pas "*infra-as-code !*" à la lecture de ces mots, revenez au début de l'article Voici un exemple de fichier, appelé *Dockerfile* :

Dockerfile

```
# On indique la base de laquelle démarrer
FROM centos:7

# On ajoute quelques meta-informations
LABEL maintainer Xebia <info@xebia-training.fr>

# On installe les dépôts EPEL pour avoir nginx
RUN yum install -y epel-release
# On installe un paquet: nginx
RUN yum install -y nginx

# On copie un fichier depuis le dossier courant
COPY index.html /usr/share/nginx/html/index.html

# On précise quel est le premier processus à être lancé
ENTRYPOINT ["/usr/sbin/nginx"]

# On passe des arguments à ce processus
CMD ["-g", "daemon off;"]
```

Cet article n'est pas l'endroit pour vous détailler les optimisations à apporter (image de base plus légère, regroupement des instructions, etc.), sachez cependant que Xebia propose des formations à ce sujet (http://training.xebia.fr/formation/formation-conteneurs-docker/).
On a donc ici la liste de toutes les **étapes nécessaires à la construction de notre image**, et c'est ensuite Docker (docker build en l'occurrence) qui se chargera de **comprendre ces instructions** et de les exécuter afin d'obtenir une image complète. Un point est souvent noté : la limitation de ce qu'on peut faire dans un Dockerfile par rapport à Ansible par exemple. Certains pourraient même avoir l'idée farfelue de construire des images Docker avec Ansible... une bonne idée qui n'en est en réalité pas une. En effet, de manière générale dans l'évolution des infrastructures et des services, nous essayons de plus en plus de garder les choses aussi simples que possibles. C'est valable aussi pour vos conteneurs : si vous avez besoin d'Ansible pour construire votre image Docker, il y a de fortes chances pour que votre conteneur et/ou votre application soit bien trop complexe et ne respecte pas un certain nombre de bonnes pratiques.

Déploiement

Une fois ces images construites, penchons-nous désormais sur leur déploiement : docker-compose ira remplacer, à petite échelle, les "docker run" manuels. A l'instar du cloud qui nous abstrait des ressources via des APIs et permet donc de faire de l'infra-as-code, c'est ici le *daemon* Docker local, exécutant différentes actions (via *containerd* qui lui fait appel à *runc*, mais c'est une autre histoire),

qui expose une API REST sur une *socket* locale :

```
$ curl --unix-socket /var/run/docker.sock http://docker/containers/json?all=1
```

La commande ci-dessus listera vos conteneurs, via un simple appel HTTP. Ainsi, il suffit d'avoir n'importe quel client à cette API pour décrire 'as-code' ce que l'on veut faire tourner.
docker-compose sera ainsi capable de traduire la description suivante en appel à cette API :

```
version: '3.3'
services:
  myservice:
    image: horgix/hello-json:latest
    ports:
      - 80:80
```

Et il en va de même pour les Orchestrateurs : qu'il s'agisse de **Kubernetes**, **Mesos** / **Marathon**, **Nomad** ou encore le mode **Swarm** de Docker, tous permettent nativement de leur décrire textuellement (JSON, YAML, HCL, peu importe) l'état que l'on souhaite afin qu'ils puissent ensuite s'assurer que cet état est respecté. C'est ce panorama qui représente les infrastructures, services et outils d'aujourd'hui : tout est contactable via une API, ce qui nous permet de composer programmatiquement plusieurs outils afin de choisir ce qui nous correspond.
Ainsi, avec la **description textuelle de l'intégralité de la chaîne** de build / déploiement et une **intégration continue** utilisant de tels outils d'infra-as-code, un simple commit sur le dépôt de mon application ira la compiler et la tester unitairement, mais génèrera également une image prête à être déployée. Cette image sera justement **déployée automatiquement** dans un environnement créé spécialement pour l'occasion, permettant de lancer les tests fonctionnels, pour au final être supprimé une fois ceux-ci exécutés. Enfin, on pourra aller jusqu'à un déploiement en production, qui entraînera lui-même une mise à jour de load balancers, une éventuelle création de certificats SSL, voire la création et l'attachement de volumes distants dans le cas d'une application avec état, le tout en interconnectant simplement des composants modernes et en décrivant ce que l'on attend d'eux textuellement.

Ce que l'on peut espérer

Avant d'arriver à la fin de cet article, voyons un aperçu de ce qui est réalisable grâce à l'infrastructure-as-code et son automatisation. Admettons que je possède un système existant, *on-premise*, et responsable d'héberger des applications Dockerisées. Mon besoin est le suivant : **je veux pouvoir faire tourner mes applications sur un nouveau serveur *bare-metal* sans avoir quoi que ce soit d'autre à réaliser que le *racker*, le brancher et l'allumer**.
Impossible ? Pas tant que ça, voyons ensemble les composants que j'aurais personnellement choisis.

- Je rack le serveur, le branche (électricité, réseau), et le démarre. Puis...
- Le serveur démarre en PXE et broadcast une demande d'IP et d'image.
- Le serveur DHCP lui répond, lui indiquant en next-server un serveur TFTP fourni par **Cobbler.**
- Cobbler, ayant au préalable des profils de machines prédéfinis (infra-as-code, souvenez-vous), lui fournit une image d'installation (ISO) ainsi qu'un *kickstart*, *preseed* ou autre en fonction de la distribution.
- Le serveur récupère l'image et démarre dessus.
- Le kickstart / preseed lui indique les informations basiques (réseau notamment) ainsi qu'une seule et unique tâche supplémentaire en dehors de l'installation de l'OS lui-même : l'installation de l'**agent de configuration** ainsi que sa configuration basique lui indiquant où se trouve son serveur central. Choisissons **Salt** pour notre exemple.
- La machine ayant fini son installation, elle redémarre.
- Lors du démarrage, l'agent Salt se lance, se connecte au master Salt, récupère la description de l'état dans lequel la machine doit se trouver et la fait converger vers cet état.
- Cet état décrit les éléments suivants : installation et configuration du *daemon* **Docker** et de l'**agent d'orchestration**, qui serait par exemple le **kubelet** si notre orchestrateur est **Kubernetes.**
- Lors du démarrage de l'agent d'orchestration, celui-ci s'authentifie auprès du *master* et s'enrôle dans le cluster.
- L'orchestrateur commence à lui envoyer des tâches/conteneurs à lancer.

Bien sûr, le point critique est la partie "le worker s'authentifie auprès du master", et c'est souvent le seul élément qui reste manuel. Il serait possible d'assurer de la signature cryptographique de bout en bout de la chaîne de provisionnement qui permettrait cette authentification, mais la complexité en serait considérablement accrue. En résumé, **nous n'avons rien fait d'autre que brancher un serveur et l'allumer : celui-ci est opérationnel et utilisé.**

Les pièges

La généricité à outrance

La volonté de réaliser de l'infra-as-code couvrant **tous** les cas est le **principal piège** : très vite, vous voudrez réaliser des rôles Ansible (par exemple) couvrant tous les cas possibles et imaginables qui seront ensuite configurables par des variables en entrée. Vous commencerez par quelques éléments, ajouterez ensuite des conditions, puis commencerez à documenter les différents cas possibles et implications, et avant que vous n'ayez pu vous en rendre compte, vous venez de créer une véritable "usine à gaz" qui sera complexe à maintenir, que vos collaborateurs seront réticents à utiliser, et qui finira par causer plus de bugs inattendus qu'elle n'en résoudra.
Un conseil : restez simple. Ne gérez que les cas dont vous avez besoin, suivez le principe **YAGNI** (You Aren't Gonna Need It - https://www.martinfowler.com/bliki/Yagni.html), et faites évoluer votre infra-as-code au fur et à mesure des cas que vous rencontrerez. Sans cette rigueur, vous vous retrouverez simplement à déplacer toute la logique de configuration dans les variables/paramètres de votre infra-as-code, tant est si bien qu'il serait plus simple pour les utilisateurs de votre code de simplement écrire le résultat eux-mêmes.

Le choix des outils

Nous vous avons présenté dans cet article des solutions principalement Open Source, par conviction mais aussi en toute objectivité : les solutions monolithiques et propriétaires visant à répondre à tous les besoins imaginables sont celles qui ont construit

les infrastructures dont nous cherchons aujourd'hui à nous abstraire. On visera aujourd'hui davantage un choix d'outils suivant la philosophie UNIX : **ne faire qu'une chose, mais la faire bien** (https://en.wikipedia.org/wiki/Unix_philosophy#Do_One_Thing_and_Do_It_Well).
La bonne nouvelle est que même les constructeurs de hardware propriétaire commencent à s'ouvrir ; au final, tant qu'un produit a une API, il sera intégrable dans une démarche d'infra-as-code. Un critère de choix sera donc l'interopérabilité de ces outils d'infra-as-code, que l'on combinera les uns avec les autres pour obtenir un tout qui réponde à un besoin précis ; un bon artisan choisit toujours les outils adaptés.
Oui, Ansible, SaltStack, Puppet et consorts ont des modules leur permettant de parler aux APIs des fournisseurs de cloud. Il est donc théoriquement possible de réaliser la même chose avec Ansible qu'avec Terraform. Mais la réalité en est bien loin : Ansible n'étant pas pensé pour, il sera nécessaire de faire toute la détection d'erreur et la reprise sur erreur soi-même, ce qui devient très vite complexe, lourd à maintenir et s'avère source d''erreur. Terraform et CloudFormation sont **faits pour abstraire la gestion de l'infrastructure au travers des APIs des fournisseurs de Cloud** et savent donc gérer les cas complexes nativement. Pour avoir réalisé la migration de pur Ansible vers Packer + Ansible, celle-ci a été salutaire d'un point de vue maintenabilité, clarté et efficacité : une demi-heure pour passer du "*boilerplate*" par Ansible à un descriptif Packer tenant en 20 lignes de JSON et appelant Ansible pour gérer de la configuration (ce pour quoi il a été conçu).

N'adapter que ses outils et non son organisation

Il est aisé de rester dans une structure organisationnelle habituelle avec développeurs et opérationnels séparés, réalisant chacun leur infra-as-code de leur côté. Mais l'infra-as-code est une parfaite **opportunité de commencer à travailler main dans la main** si ce n'est pas déjà le cas : saisissez-la !
Développeurs : n'hésitez pas à montrer vos définitions d'infra-as-code, que ce soit vos dépendances, votre Dockerfile ou vos rôles Ansible à vos Ops, ils seront ravis et peut-être même impressionnés. Opérationnels : ne fermez pas votre infra-as-code à vous seul, incitez les développeurs à vous envoyer leurs propres Pull/Merge Requests que vous pourrez alors review ensemble.

Conclusion

Pour conclure ce dossier sur l'*infrastructure-as-code*, nous souhaitons vous adresser un "take away" sous forme de quelques objectifs à atteindre si vous souhaitez faire de l'infra-as-code, et le faire bien :

- **Vous devez être en mesure d'open-sourcer la description de l'infrastructure**. Ceci ne signifie pas forcément que vous devez le faire, mais vous devez vous poser la question "*Si je publiais publiquement le code de mon infrastructure là maintenant, cela poserait-il un problème ou un risque de sécurité ?*". Si la réponse est oui, il y a des chances pour que vous veniez de trouver votre prochain axe d'amélioration !
- **Vous devez pouvoir recréer votre infrastructure ailleurs en une seule commande.** Pour un peu plus de tolérance, disons 2 ou 3. Si vous ne pouvez pas, deux possibilités : soit vos outils manquent d'interopérabilité et vous devriez les connecter les uns aux autres, soit votre infra-as-code a toujours besoin d'entrées manuelles, ce qui est tout l'inverse de ce que l'on cherche à réaliser.
- **Vos serveurs peuvent se voir assigner un UUID en lieu et place de leur hostname habituel.** Vous devez être capables de raisonner en termes de rôles/groupes pour vos serveurs et non en termes de serveurs eux-mêmes.
- **Il doit être possible de comprendre votre infrastructure en clonant un ou plusieurs dépôts Git et en lisant ce qui y est contenu.** L'intégralité de votre infrastructure doit être décrite textuellement et versionnée. Demain, vous devez pouvoir recréer tout le SI de votre entreprise à partir de ces descriptions et des backups de bases de données en un claquement de doigts.

Enfin, nous souhaiterions apporter un dernier mot : **ces cas peuvent vous sembler idéalistes, inatteignables, mais il n'en est rien**. Ce sont des pratiques que nous mettons fréquemment en place chez Xebia et qui assurent à nos projets et leur infrastructure une stabilité et une facilité de maintenance. Ceci nous satisfait et satisfait nos clients, facilitant d'autant plus les passages de compétences. Vous **devez** être en mesure de tout gérer via de l'infrastructure-as-code. Si un élément n'est pas gérable automatiquement, c'est probablement qu'il nécessite d'évoluer pour qu'il le soit. N'hésitez pas à vous lancer, à être ambitieux et à avoir comme objectif de **tout** automatiser et de **tout** gérer comme du code ! •

Christophe PICHAUD
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com | www.windowscpp.com

NeOS-SDI makes IT work

Migrer son code C/C++ en 64-bits

De nos jours, Windows est installé en 64 bits : on dit aussi mode x64 ou architecture amd64. Dans les fichiers projets, on nous propose de gérer cette cible x64 mais il y a de nombreux trucs et astuces à connaître lors du passage. Ce n'est pas si automatique que de juste recompiler les sources de vos applications et dll… Nous allons balayer les différents éléments avec le plus de détails possibles. Objectif : « Ça build ! »

Les outils nécessaires

Lorsque vous installez Visual Studio, vous avez le choix entre plusieurs SDK et plusieurs compilateurs. Actuellement, CL.EXE, le compilateur Microsoft est proposé en 4 versions :

- cl.exe 32 bit pour X86,
- cl.exe 64 bit pour X64,
- cl.exe 32 bit et 64 bit pour ARM.

Les grands changements

Le type int et long sont des valeurs 32-bit. Le type size_t est une valeur 64-bit. Voici le détail :

Scalar Type	C Data Type	Storage Size (in bytes)	Recommended Alignment
INT8	**char**	1	Byte
UINT8	**unsigned char**	1	Byte
INT16	**short**	2	Word
UINT16	**unsigned short**	2	Word
INT32	**int, long**	4	Doubleword
UINT32	**unsigned int, unsigned long**	4	Doubleword
INT64	**__int64**	8	Quadword
UINT64	**unsigned __int64**	8	Quadword
FP32 (single precision)	**float**	4	Doubleword
FP64 (double precision)	**double**	8	Quadword
POINTER	*****	8	Quadword
__m64	**struct __m64**	8	Quadword
__m128	**struct __m128**	16	Octaword

La gestion des chaînes de caractères est plus délicate. Faisons un petit rappel sur les définitions Windows. Il existe CHAR, WCHAR, LPSTR, LPCSTR, LPTSTR, LPCTSTR. Voyons les définitions Windows et leurs équivalents que votre projet supporte l'Unicode ou pas.

```
// Generic types

#ifdef UNICODE
   typedef wchar_t TCHAR;
#else
   typedef unsigned char TCHAR;
#endif

typedef TCHAR *LPTSTR, *LPTCH;

// 8-bit character specific

typedef unsigned char CHAR;
typedef CHAR *LPSTR, *LPCH;

// Unicode specific (wide characters)

typedef unsigned wchar_t WCHAR;
typedef WCHAR *LPWSTR, *LPWCH;
```

Si le projet supporte Unicode, la gestion des chaînes est assurée via wchar_t pour TCHAR. Ce qu'il faut retenir de ces déclarations c'est que TCHAR est un allié utile. En effet, TCHAR via tchar.h va nous permettre de faire un code générique dans la gestion des chaînes de caractères. Pour déclarer une chaîne, on utilise la définition TCHAR :

```
TCHAR sz[255];
```

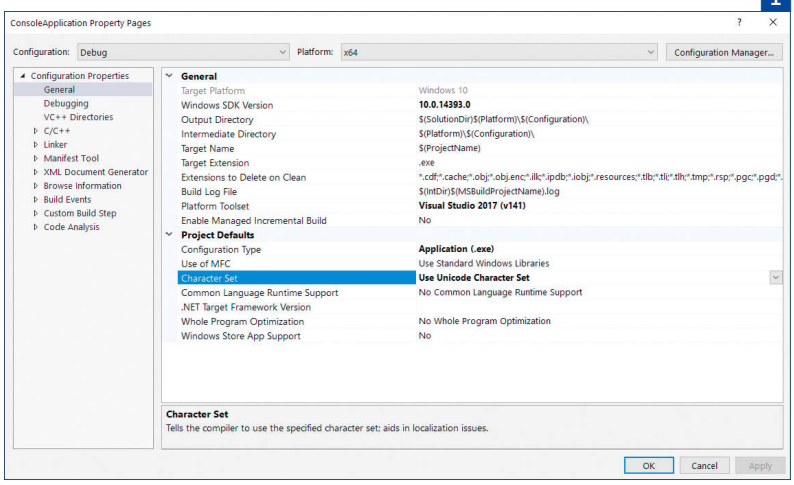
TCHAR vaudra soit char ou wchar_t. Comment le savoir ? Tout dépend si votre projet est Unicode ! Si vous êtes en « Character Set » = « Use Unicode Character Set » alors votre projet va définir la constante UNICODE au niveau du préprocesseur. Regardons les propriétés du projet où cela est défini : 1

Pour être transparent, je vous engage à coder de manière multiplateforme. Pour cela, il faut une table de correspondance de routines C. https://msdn.microsoft.com/en-us/library/tsbaswba.aspx

En voici un extrait : 2

Si vous faites du TCHAR, vous utilisez la routine générique. Et suivant si UNICODE est définie ou pas, le nom de la fonction est étendu en nom voulu. Exemple de code à migrer :

1



```
char sz[255];
sprintf(sz, "Elapsed ms since Windows is up: %ld ms\n", GetTickCount());
printf(sz);
```

Pour que ce code compile en x86 et en x64, il faut prendre des précautions et obtenir la correspondance de sprintf et printf en mode TCHAR.h :

Generic-Text Routine Mappings

TCHAR.H routine	_UNICODE & _MBCS not defined	_MBCS defined	_UNICODE defined
_stprintf	sprintf	sprintf	_swprintf
_stprintf_l	_sprintf_l	_sprintf_l	__swprintf_l

Generic-Text Routine Mappings

TCHAR.H routine	_UNICODE & _MBCS not defined	_MBCS defined	_unicode defined
_tprintf	printf	printf	wprintf

Ainsi, nous savons quels sont les noms des fonctions à employer :

```
#include <tchar.h>

TCHAR sz2[255];
_stprintf(sz2, _T("Elapsed ms since Windows is up: %ld ms\n"), GetTickCount());
_tprintf(sz2);
```

Comme vous pouvez le constater, les chaînes sont bordées par _T(" "). En mode Not Set ou MBCS, _T ne vaut rien alors qu'en UNICODE, il vaut L. C'est comme ça que l'on désigne une chaîne Unicode.

Autre solution ISO : la STL

Il est possible de gérer le type chaîne de caractères via la STL du C++ : la Standard Template Library. Il suffit de faire #include <string> et de déclarer soit des std::string ou des std::wstring. Ce sont soit des chaînes de type char ou de type wchar_t. Tout dépend si on veut coder pour Windows ou multiplateforme. Pour être complètement ISO, je vous engage à utiliser cette solution à base de STL. Le standard ISO, c'est le langage et la STL. Sachez qu'entre experts, nous n'avons pas tous le même point de vue. Mon pote Alain Z, de Redmond recommande de coder en wchar_t. En fait, moi je vois deux modes. Suivant si on fait une dll ou suivant si on fait un exe, suivant si on attaque des API Windows, suivant si on fait du Boost, un ensemble de librairies C++ très connues disponible sur Boost.org. Si on fait du Boost, on va se retrouver à faire du string ou du wstring à tous les étages (threads, filesystem, regexp, etc). Voici ce que cela donne avec la STL :

```
#include <string>
#include <iostream>

std::wstring sz3 = _T("hello world !");
std::wcout << sz3 << std::endl;
```

Les conversions std::string & std::wstring

Si vous utilisez les API Windows, vous serez amené à utiliser TCHAR. Si vous avez du code externe, par exemple, qui utilise la STL et std::wstring il va falloir faire des correspondances. Comment fait-on ? Avec la STL ce n'est pas difficile ; on peut revenir au type de base via la méthode c_str() :

```
std::wstring s1 = L"UNICODE direct";
const wchar_t * ws1 = s1.c_str();

std::string s2 = "Hello world";
const char * ss2 = s2.c_str();
```

En utilisant le ctor de wstring on peut passer d'un string à wstring facilement et vice-versa :

```
std::string a1 = "A1";
std::wstring u1 = _T("U1");

std::wstring u2(a1.begin(), a1.end());
std::wcout << u2 << std::endl;

std::string a2(u1.begin(), u1.end());
std::cout << a2 << std::endl;
```

Ce genre d'opérations n'est pas coûteux. Pour évoluer dans une situation confortable, je choisis la STL et je fais des allers-retours de temps en temps avec les TCHAR quand il faut faire des appels avec les API Windows. Si je fais une GUI en MFC, j'utilise les classes CString. Vous allez me dire : « mais c'est le "bordel", on utilise tout ??? ». Vous avez raison mais voici mon explication. Certaines API prennent des types ; si on passe son temps à convertir, on ne s'en sort plus… C'est déjà assez complexe par exemple quand en COM on doit manipuler des BSTR ou des VARIANTS… Bref, utilisez ce qui est naturel.

Autre solution Windows via les MFC et ATL/MFC

Il est aussi possible de prendre une classe générique CString gérée par la partie commune ATL/MFC. Cette classe CString existe en plusieurs variantes : CStringT, CStringA, CStringW. Simple à comprendre si vous avez lu l'article depuis le départ… Au milieu de ce code vous pouvez voir les fonctions A2T et T2A qui sont bien pratiques. Pour les utiliser, il faut déclarer la macro USE_CONVERSION juste derrière le { d'entrée dans la fonction.

```
CString cs1 = sz;
CString cs2 = s1.c_str();
CString cs3 = A2T(s2.c_str());
CStringW cs4 = sz;
CStringW cs5 = s1.c_str();
```

Generic-text routine name	SBCS (_UNICODE & MBCS not defined)	_MBCS defined	_UNICODE defined
_cgetts	_cgets	_cgets	_cgetws
_cgetts_s	_cgets_s	_cgets_s	_cgetws_s
_cputts	_cputs	_cputs	_cputws
_fgettc	fgetc	fgetc	fgetwc
_fgettchar	_fgetchar	_fgetchar	_fgetwchar
_fgetts	fgets	fgets	fgetws
_fputtc	fputc	fputc	fputwc
_fputtchar	_fputchar	_fputchar	_fputwchar
_fputts	fputs	fputs	fputws
_ftprintf	fprintf	fprintf	fwprintf

2

```
CStringA cs6 = s2.c_str();
CStringA cs7 = T2A(s1.c_str());
```

Il existe des variantes :

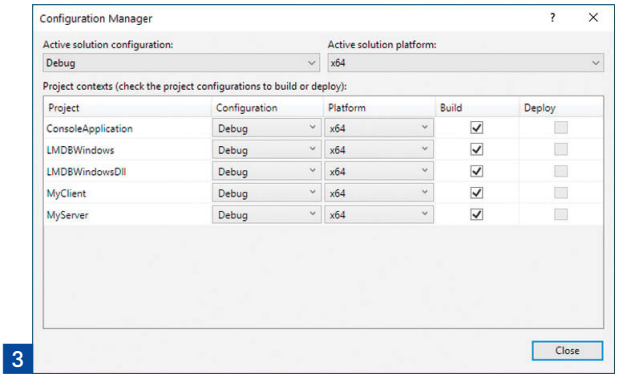
- A2T et T2A
- W2T et T2W
- A2W et W2A

Compiler directive in effect	T becomes	OLE becomes
none	A	W
_UNICODE	W	W
OLE2ANSI	A	A
_UNICODE and OLE2ANSI	W	A

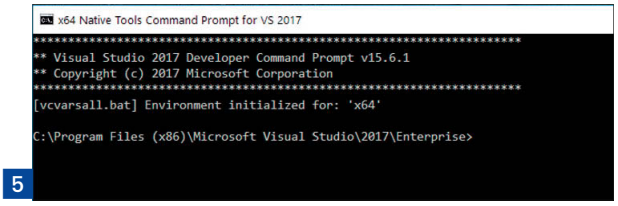
Dans le nom des macros, le type de chaînes sources est à gauche et le type de chaînes destinations est à droite. A veut dire LPSTR, T pour LPTSTR et W pour LPWSTR.

Le projet C++ x64 type

Tout se passe dans la boîte de paramétrages du projet. Il faut définir une plateforme x64 via le Configuration Manager si elle n'existe pas… 3



3



5

Nommage des modules

Il faut une certaine rigueur dans le nommage des modules. Boost est un modèle à suivre.

On connaît les informations suivantes rien qu'au nom du module :

- Est 32 bit ou 64 bit,
- S'il est debug ou pas,
- S'il est shared ou static,
- La version du Visual C++ avec lequel il est compilé,
- La version de la librairie. 4

Rappel sur les dépendances

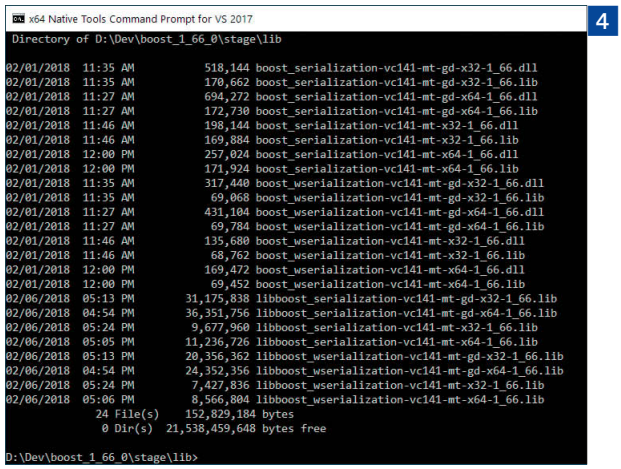
Les modules doivent tous être compilés en x64. On ne mixe pas x86 et x64 sinon ça plante ! On build soit en release soit en debug mais on ne mixe pas !

Compilation en ligne de commandes

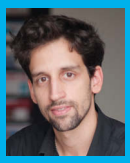
Il faut lancer le raccourci « x64 Native Tools Command Prompt for VS 2017 » et voilà : 5

Conclusion

Le passage en 64-bit via la recompilation des sources peut-être un travail coûteux et qui prend du temps. Tout dépend de la qualité du code. La gestion des chaînes de caractères peut-être le frein à ce passage. Avec les règles décrites dans cet article, on possède les trucs et astuces pour s'en sortir sans trop forcer. •



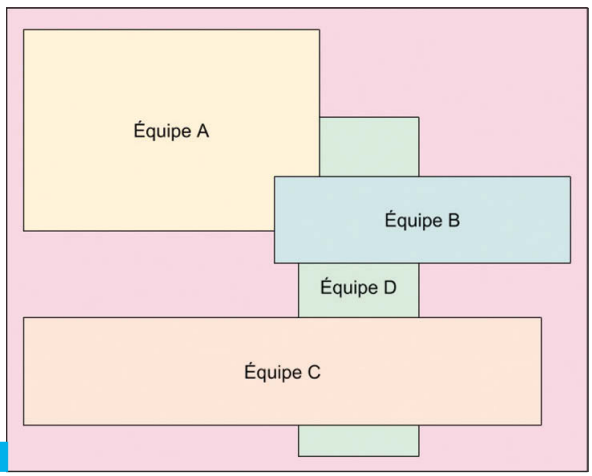
4



Rudy RIGOT
Ingénieur logiciel et startup mentor pour TechStars, actuellement en poste chez Kenna Security à Chicago, cofondateur de l'école Holberton School à San Francisco, auteur du livre Programmer en Go : Pourquoi ? Comment ? paru aux éditions D-BookeR (https://www.d-booker.fr). Twitter : @rudyrigot

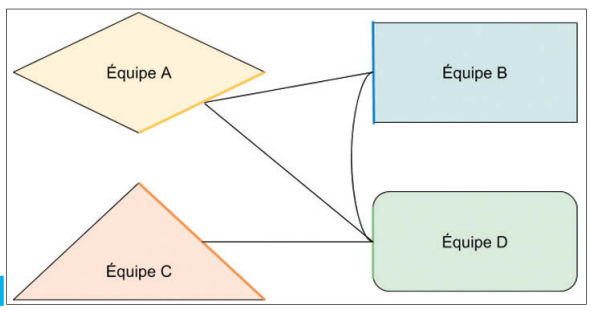
Go, le langage ultime des micro-services ?

Si les architectures de micro-services et orientées services ne sont pas récentes, un œil observateur sur les retours d'expériences du marché aura pu remarquer une accélération de la mise en avant de Go dans les billets de blog d'ingénierie sur ce type d'architecture. Certains produits influents et à grande échelle tels que Uber, Twitch, Dropbox, Heroku ou bien sûr Google (1) annoncent désormais faire de Go le fer de lance de leur stratégie technique pour leurs micro-services.
Pourquoi cette accélération du marché autour des micro-services en Go ? Cet intérêt est-il fondé ? Les solutions apportées par Go correspondent-elles aux problèmes posés par les micro-services ? Tentons une analyse objective des besoins pour mieux comprendre d'où le marché vient, et en déduire potentiellement où il va.



1

Architecture monolithique : l'implémentation technique est similaire pour toutes les équipes, chaque équipe a la possibilité d'impacter l'implémentation des autres équipes. Le fruit du travail de toutes les équipes est déployé simultanément.



2

Architecture de micro-services : les équipes ne peuvent utiliser les services des autres équipes que via une interface stricte, et ne peuvent pas influencer leur implémentation. L'implémentation technique de chaque service est potentiellement très différente, et chacun est déployé séparément.

Pourquoi les micro-services ?

Comme toute solution technique, la technologie Go et les architectures de micro-services sont utilisées pour résoudre avant tout des problèmes métier. Gardons en tête que l'objectif principal des architectures de micro-services est de solutionner les problèmes rencontrés par les équipes d'ingénierie larges. Cela permet un meilleur découpage de la propriété technique et métier de chaque équipe, tout en limitant les effets de bord des équipes les unes sur les autres.

L'alternative est connue, puisque c'est une réalité aujourd'hui pour les produits en ligne dont la phase de croissance a eu lieu avant que les architectures à base de monolithes ne soient remises en question. Prenons par exemple le monolithe Salesforce, créé en 1999, qui contient aujourd'hui 8,5 millions de lignes de code et auquel 800 ingénieurs contribuent chaque jour. La mise à jour occasionnelle d'un environnement local prend généralement environ une heure ; envoyer des changements de code prend aussi généralement une heure ou plus, et échoue souvent. Les systèmes d'intégration continue sont tellement surchargés qu'il n'est pas rare d'être notifié plusieurs jours après une soumission de code, alors qu'elle a déjà été acceptée, qu'elle s'avère casser un test existant ! Cette réalité est bien vivace pour moi, puisque j'ai personnellement contribué à ce monolithe pendant plus d'un an en 2016-2017. 1 2

Ainsi va la vie pour les gros produits monolithiques, et il est désormais un fait notoire que les startups en phase de forte croissance gagnent souvent à diviser leurs services en plusieurs applications distinctes avec un couplage le plus léger possible pour éviter d'arriver à ce résultat. Chaque équipe s'engage ainsi auprès des autres équipes uniquement sur la manière d'utiliser leurs services, et maîtrise à son gré l'implémentation interne ; et ainsi, chaque équipe gère indépendamment sa propre dette technique.

Salesforce a parfaitement conscience de ce nouveau consensus de l'industrie et tente aujourd'hui d'orienter son monolithe vers ce modèle, mais avec difficulté, car le retard s'est accumulé. À l'opposé, le risque existe aussi pour un produit de s'orienter trop tôt vers des micro-services, car ils apportent aussi leur lot de problèmes. Par exemple au niveau de leur observabilité, de l'initialisation de la communication inter-service et leur capacité à être gérés en production par la même équipe pour plusieurs équipes de développement. Ce dernier problème est toutefois largement amélioré par la philosophie DevOps et les containers applicatifs tels Docker.

(1)
Dropbox: https://blogs.dropbox.com/tech/2014/07/open-sourcing-our-go-libraries/
Uber: https://eng.uber.com/go-geofence/
Google: https://talks.golang.org/2012/splash.article
Heroku: https://blog.golang.org/go-at-heroku
Twitch: https://blog.twitch.tv/gos-march-to-low-latency-gc-a6fa96f06eb7

Une autre question se pose : à quel point un service doit-il être micro ? Quelle taille de service est idéale ? L'industrie semble s'entendre sur le fait que le mot « micro » n'est pas approprié, dans la mesure où l'objectif n'est pas de les avoir les plus petits possibles ; mais au-delà de ça, on observe diverses approches. Dans son livre « Building microservices » publié en 2015 aux éditions O'Reilly, Sam Newman établit comme critère que pour conserver l'adaptabilité d'un service aux besoins métier et aux évolutions techniques, chaque service doit pouvoir être entièrement reconstruit en moins de deux semaines ; mais cette définition est très controversée.

Les requis techniques pour implémenter un micro-service

Sur la base de ces objectifs métiers, tentons de dresser une liste des requis techniques pour une technologie idéale sur laquelle construire un micro-service typique :

- Un service doit pouvoir être entièrement remplacé avec un investissement raisonnable. —> La technologie doit permettre de publier un tout nouveau service en très peu de temps et doit veiller à la productivité du développeur.
- La base de code restera relativement petite à long terme. —> La technologie ne doit pas imposer l'emploi de design patterns, contrairement à un grand nombre de frameworks riches qui doivent couvrir un plus large spectre de besoins métiers à partir d'une même base de code.
- Il peut y avoir une forte latence pendant que les services s'attendent mutuellement. —> La technologie doit nativement très bien gérer les entrées-sorties de manière asynchrone.
- L'interface du service peut se retrouver très exposée à Internet sans pouvoir bénéficier d'efforts de sécurité mutualisés. —> La technologie doit proposer nativement une sécurité applicative solide.
- Optionnellement, puisque de la performance sera perdue en latence réseau, il pourrait être bienvenu d'avoir une performance interne au service optimale. —> La technologie doit être performante en termes de consommation des ressources. Elle doit également offrir un modèle de concurrence avancé pour accéder à ces ressources de manière optimale, si elles peuvent être accédées en parallèle.
- On observe que 92% des problèmes en production d'outils serveur web typiques viennent d'erreurs non-gérées (source : https://github.com/ardanlabs/gotraining/tree/master/topics/go). —> La technologie doit offrir une gestion d'erreur stricte et informative.
- L'ensemble du système est composé de plusieurs applications distinctes qui seront déployées séparément. —> Le déploiement de chaque service doit être aussi trivial que possible.
- La seule partie applicative en commun à tous ces services est celle qui leur permet de converser ensemble —> La librairie serveur HTTP de la technologie doit être riche, et proposer un support solide des protocoles de communication modernes (comme gRPC), même s'il vient de la communauté.
- Et comme d'habitude, pour des raisons évidentes, la technologie doit être facile à apprendre et à maintenir.

JavaScript, premier langage inattendu des architectures micro-services

La technologie qui a initialement le plus bénéficié de l'avènement des micro-services comme solution technique au problème des équipes en croissance est sans doute Node.js. Ce n'est pas particulièrement surprenant, car cette technologie satisfait les points critiques de la liste ci-dessus souvent mieux que les technologies concurrentes.

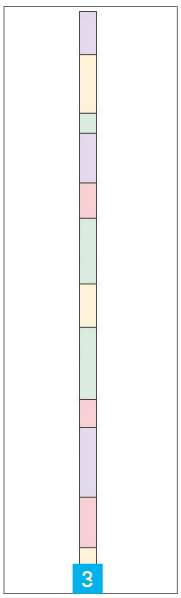
- Entièrement remplaçable—> le code basique et la configuration nécessaires à un projet Node.js est très peu verbeux, le projet peut être démarré et itéré très rapidement.
- Favorise une petite base de code, avec peu de design patterns —> il est conçu initialement pour du script dans le navigateur, donc par définition son utilisation est optimisée pour les petites bases de code.
- Gère très bien les entrées/sorties —> c'est le coup de génie de Ryan Dahl, qui a créé Node.js parce qu'il voulait sensibiliser au besoin d'avoir des entrées/sorties non-bloquantes ; les callbacks étant déjà culturellement populaires dans la communauté JavaScript, le concept allait être plus simple à expliquer.
- Facile à apprendre —> encore mieux : le langage est déjà fortement connu d'une grande majorité de développeurs, puisqu'il est incontournable dans le navigateur.

Rapidement sont apparues des librairies enrichissant Node.js d'une API serveur HTTP solide et riche (comme Express.js), ainsi que d'autres pour les divers protocoles de communication.

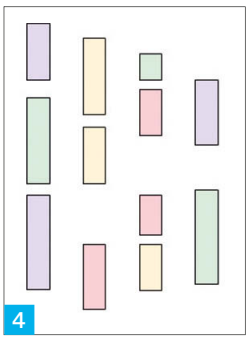
Mais il n'est pas possible de résoudre avec JavaScript les autres requis que nous avons listés ci-dessus :

- Sa sécurité est relativement problématique, puisque sa culture favorise les micro-dépendances ; donc chaque service, même manipulant des données extrêmement sensibles, dépend en pratique de code écrit et maintenu par des gens extérieurs à l'organisation et exécuté côté serveur. . De plus il a une forte dépendance avec des binaires écrits en C, dont l'absence de mémoire protégée provoque des erreurs mémoire qui sont une cause majeure des vulnérabilités modernes (2).
- Sa performance en consommation interne de ressources est peu optimale, car c'est par définition un langage interprété, et à fil d'exécution simple (ne pouvant utiliser qu'un seul cœur de processeur simultanément, donc une seule tâche est réellement exécutée à la fois). Aussi, son modèle de concurrence est relativement faible : il est aisé de créer plusieurs tâches simultanées en lançant divers callbacks, mais il est très difficile de synchroniser ces tâches ou de les faire communiquer ensemble. 3 4

Quant à sa stratégie de déploiement (fichier source par fichier source, typique d'un langage interprété) et sa stratégie de gestion d'erreur (à base d'exceptions), elles sont représentatives de la plupart des langages de l'industrie.



Les environnements d'exécution à fil d'utilisation simple comme ceux de JavaScript donnent l'illusion d'une exécution simultanée, mais une seule tâche peut être exécutée à la fois. Ils arrivent à ce résultat en donnant la main aux tâches prêtes à être exécutées lorsque la tâche en cours se retrouve bloquée (par exemple, à cause d'une attente d'entrée/sortie). Lorsque l'une des tâches prend du temps de CPU à exécuter, les autres tâches attendent.



Les environnements d'exécution à fils d'utilisation multiples comme ceux de Go utilisent tous les cœurs de processeurs (quatre, dans le schéma ci-dessus) pour réellement exécuter les tâches en parallèle, même si l'une d'entre elle a de longues opérations CPU.

(2)
Source : https://www.vvdveen.com/memory-errors/
Une autre source, limitée au cas d'utilisation des navigateurs : https://bentrask.com/?q=hash://sha256/c8909ef4e934d5954f1ef8c8

Comparaison des requis techniques des micro-services avec les solutions de Go

Si l'invention de JavaScript est très antérieure aux micro-services et explique pour beaucoup ses quelques imperfections au regard de ce besoin, Go n'a initialement pas été créé pour les micro-services non plus. L'objectif initial était de concevoir un langage pour remédier aux carences que Google rencontrait avec ses monolithes web à grande échelle. La légende voudrait que Go ait été imaginé pendant les 45 minutes nécessaires à la compilation d'un monolithe en C++.

Cela explique pour beaucoup la proximité de Go avec les besoins des micro-services malgré tout, puisqu'il s'agit aussi de services web, et à grande échelle ; simplement les bases de code sont beaucoup plus petites que prévu. Mais cela explique aussi pour beaucoup les problèmes que Go résout mal, tels que la gestion de dépendances (parce que l'ensemble du code interne de Google est sur la même branche du même système de gestion de source) et la réutilisabilité entre projets (notamment observée par l'absence très controversée de programmation générique).

Si l'on reprend les points forts de JavaScript au regard des besoins listés plus haut, on s'aperçoit rapidement que Go y satisfait de manière au moins équivalente, sinon mieux :

- Entièrement remplaçable—> un projet est grosso modo aussi facile à démarrer en JavaScript et en Go, et certainement plus simple à itérer en Go, le langage et l'outillage ayant une orientation plus forte vers l'expérience développeur.
- Architecture simple —> les deux plateformes sont aussi flexibles l'une que l'autre au regard des design patterns.
- Gestion des entrées/sorties —> les deux les gèrent de manière similaire, il y a aussi des closures et des callbacks en Go, mais aussi d'autres manières de rendre une opération non-bloquante.
- Facile à apprendre —> les retours d'expérience faisant l'éloge de la courbe d'apprentissage de Go ne manquent pas, notamment grâce à son nombre de mots-clés limité à 25 et sa politique de refuser tout ajout de fonctionnalités non susceptibles d'être utiles à tous, à sa standardisation très forte garantissant que tous les développeurs Go écrivent un code similaire, à son orientation forte vers l'expérience développeur, etc.

Quant aux autres points que nous avons listés et pour lesquels JavaScript était soit en décalage, soit standard, Go propose des solutions plus adaptées :

- Sécurité —> à l'extrême inverse des micro-dépendances de JavaScript, Go propose une librairie standard riche et décourage fortement les dépendances vers du code tiers. Aussi, ni C ni aucun langage à mémoire non-protégée n'est mis à contribution, évitant à coup sûr les problèmes de sécurité liés aux fuites mémoire : le compilateur Go lui-même est écrit... en Go !
- Performant en consommation de ressources —> peu surprenant, puisque le langage était destiné à soutenir des monolithes auparavant écrits en C++. Comparé à JavaScript, Go est un langage compilé et à fil d'exécution multiple (le même programme peut ordonnancer ses tâches entre les différents cœurs d'un microprocesseur, aucune tâche ne peut bloquer toutes les autres). Il offre ainsi un modèle de concurrence très puissant à moindre coût, permettant aux tâches de

UN EXEMPLE DE MICRO-SERVICE EN GO

Ci-dessous est un exemple typique de micro-service en Go appelant une seule fonction fournie par un package dédié pour notre logique métier (election.Winner). Tout le reste ici est fourni par la librairie standard. Vous remarquerez que le langage est peu verbeux pour l'exposition d'une route et l'adaptation du comportement du service en cas de souci. De plus, notez que si vous venez d'un autre langage à syntaxe impérative, vous n'aurez pas besoin d'explication pour comprendre ce qui se passe à chaque ligne.

```
package main

import (
  "fmt"
  "log"
  "net/http"

  "github.com/foo/bar/election"
)

// Winner is an HTTP handler to write in the response the winning politician
func Winner(w http.ResponseWriter, r *http.Request) {
  winner, err := election.Winner()
  if err != nil {
    w.WriteHeader(http.StatusInternalServerError)
    w.Write([]byte(err.Error()))
    return
  }

  fmt.Fprintf(w, "%s", winner.String())
}

func main() {
  http.HandleFunc("/winner", controller.Winner)
  log.Fatal(http.ListenAndServe(":8080", nil))
}
```

se synchroniser et communiquer entre elles de manière riche.

Gestion d'erreur —> Comme c'est aussi un problème courant dans les monolithes web, Go met le paquet en forçant le développeur à gérer chaque erreur possible (le programme ne compilera pas sinon), quitte à entraîner une certaine verbosité sur le sujet.

- Déploiement —> Le besoin existant aussi pour les monolithes web, l'approche de Go est là encore extrême, car le résultat d'une compilation Go est un seul fichier exécutable en langage machine, sans aucune dépendance. La méthode de déploiement est donc d'envoyer ce fichier vers le serveur, puis de l'exécuter, pouvant ainsi mettre à jour toute la couche applicative en une opération. Pour rendre ce déploiement encore plus facile, il est possible de cross-compiler un programme d'un système d'exploitation vers un autre : par exemple, de compiler sur son ordinateur de développement sous macOS ou Windows, et en obtenir localement un exécutable en langage machine pour Linux, à envoyer directement sur un serveur Linux.
- Librairies web —> puisque le cas d'utilisation initial était le monolithe web, l'API serveur HTTP faisant partie de la librairie standard est particulièrement riche, nativement plus ou moins aussi riche que ce qui est fourni par la librairie tierce Express.js pour Node.js. Aussi, chaque protocole de communication (comme gRPC) a typiquement une librairie qui en est l'autorité canonique solide.

En conclusion

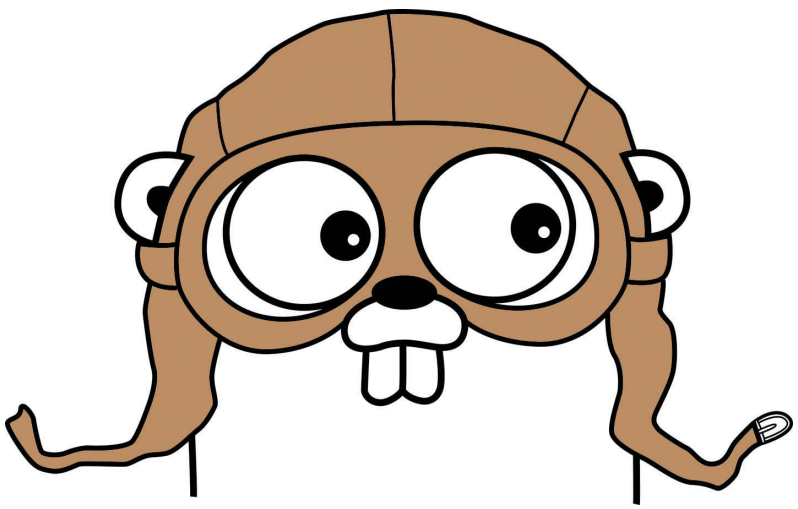
Dans la mesure où Go répond non seulement aux requis couverts par Node.js mais aussi à d'autres, et que Node.js figure déjà en très bonne position pour développer ce type d'architecture, on peut raisonnablement conclure que le succès grandissant de Go dans le domaine est justifié.

Comme les micro-services sont, par définition, simples à remplacer en cas d'évolutions technologiques, il est facile de prévoir une réécriture progressive des micro-services existants en Node.js vers Go, en sus des nouveaux services créés. Il n'est pas rare d'ores et déjà de lire des retours d'expérience sur ce type de migration (par exemple, le retour très riche et mesuré de Digg, aux résultats impressionnants : http://blog.digg.com/post/141552444676/making-the-switch-from-nodejs-to-golang).

Cela dit, comme pour toute solution technologique, le besoin métier ou le paysage technologique peuvent tout à fait apporter de bonnes raisons de préférer d'autres langages à Go, même pour le cas d'utilisation d'un micro-services. Quelques exemples :

- Si votre service n'a pas lieu d'être utilisé par des clients (par exemple, c'est un service interne), alors sa performance et sa rigueur contre les erreurs importent sûrement moins ; et si en plus, il a peu de chance de changer dans le temps, alors sa maintenabilité importe peu aussi. Auquel cas, le besoin peut certainement être rempli par n'importe quel langage, et il n'est sans doute pas optimal d'en apprendre un nouveau.
- Si votre service requiert une énorme capacité de mémoire, alors il s'avérera essentiel de libérer le plus rapidement possible la mémoire, auquel cas le ramasse-miette de Go risque de se trouver en travers de votre chemin. C'est un cas extrêmement marginal, qui ne s'applique réellement qu'aux produits à très grande échelle. Mais il peut vraiment se présenter : par exemple, Dropbox a décidé de réécrire de Go vers Rust les parties de son service de transformation d'images où l'utilisation de la mémoire est la plus dynamique parce que Rust n'a pas de ramasse-miettes (cela dit, le micro-service en lui-même reste principalement implémenté en Go et embarque juste quelques morceaux de code machine écrits en Rust).
- Si l'entreprise a déjà largement investi dans l'utilisation d'une plateforme particulière, alors l'utilisation d'autres technologies peut ajouter des difficultés légitimes. C'est le cas chez Salesforce, par exemple, qui a tellement investi dans ses outils pour la JVM qu'il est aujourd'hui impossible de déployer autre chose sur ses serveurs de production (des projets sont en cours pour solutionner ce problème).
- Et bien sûr, en regardant vers un avenir hypothétique et certainement plus lointain, gardons en tête que Go n'a pas été créé pour solutionner le besoin des micro-services, et a donc des choix de conception propres à son objectif initial d'être un langage pour monolithe. Par exemple, le langage fait des choix forts pour conserver une rapidité extrême de compilation pour des gros volumes de code, ce qui n'est pas utile dans le contexte des micro-services. Peut-être existera-t-il dans le futur une nouvelle technologie qui sera créée précisément pour répondre à ce besoin, et sera encore plus appropriée? Aucune technologie ne semble prendre cette direction aujourd'hui, mais c'est certainement une possibilité envisageable.

Actuellement, cela dit, si vous travaillez ou souhaitez travailler pour des produits en phase de croissance, alors l'architecture micro-services croisera certainement votre chemin, et il y a de fortes chances que Go finisse par faire partie de votre quotidien, d'une manière ou d'une autre. •



Loïc Devulder,
Senior QA Engineer chez
SUSE

openQA, après la théorie, la pratique ! Partie 2

Après avoir vu comment fonctionne l'outil openQA le mois dernier, nous allons maintenant voir comment le mettre en oeuvre avec un cas concret.

Pré-requis

Afin de pouvoir utiliser openQA il faut que le produit soit installé (ça coule de source, mais c'est mieux en le disant !). Il est possible de l'installer sur un PC sous Linux (openSUSE de préférence ;-)). 4Go de mémoire avec un dual-core suffisent à faire tourner openQA pour notre test, mais idéalement 8Go de RAM et un quad-core permettent d'être tranquille.

L'installation en elle-même n'est pas trop compliquée. Je ne vais pas la décrire dans cet article principalement car le produit évolue rapidement, et ce que je pourrais écrire risque de ne plus être d'actualité au moment où vous lirez ces lignes. Je vous invite donc à suivre la procédure d'installation, très bien décrite, à l'adresse suivante : https://github.com/os-autoinst/openQA/blob/master/docs/Installing.asciidoc.

Pensez également à lire ce document en pré-requis : https://github.com/os-autoinst/openQA/blob/master/docs/GettingStarted.asciidoc, cela vous permettra d'appréhender l'outil. N'oubliez pas non plus de récupérer les codes des tests openSUSE ainsi que les *needles* correspondants : https://github.com/os-autoinst/openQA/blob/master/docs/GettingStarted.asciidoc#get-testing.

Sans être un expert de Perl (ce n'est pas mon cas), une connaissance de ce langage est un avantage pour comprendre le code des tests.

C'est également implicite, mais une connexion internet est également nécessaire, une ligne ADSL 8Mb suffit.

Bon alors, qu'est-ce qu'on teste ?

J'aurais pu facilement choisir de tester l'installation d'un OS, la partie High Availability (mon domaine) ou bien un site Web. Mais pour une première approche je voulais quelque chose de plus fun qui n'avait pas encore été fait sous openQA (à ma connaissance). J'ai donc choisi de tester un jeu ! Mais bon, tester un jeu récent, souvent en 3D, nécessite un PC puissant ainsi qu'une carte graphique adéquate, ce n'est clairement pas ce type de jeu que j'ai choisi.

Etant un fan des jeux "point n' click" des années 80, c'est donc tout naturellement que j'ai choisi de tester l'émulateur **ScummVM** (http://www.scummvm.org). Il fallait également trouver un jeu, idéalement en Open Source ou en téléchargement libre. Ça tombe bien, Le jeu culte **Beneath A Steel Sky** (**BASS**) est justement librement téléchargeable sur le site de **ScummVM** !

Note

Les codes sources des tests ScummVM ainsi que les needles sont disponibles sur GitHub : https://github.com/ldevulder/openqa-scummvm.

Tout d'abord il faut configurer un minimum openQA

Après l'installation il faut maintenant configurer openQA :

- le type de serveur via le menu "**Machines**",
- le type d'OS via le menu "**Medium types**",
- la définition des tests via le menu "**Test suites**",
- les liens tests/OS/serveur via le menu "**Job groups**".

Type de serveur

Le type de serveur définit l'architecture matérielle : processeur, mémoire, taille du disque, type de disque, etc. Notre test ne nécessitant pas beaucoup de ressources nous allons utiliser quelque chose de classique : un processeur de type x86_64 avec une configuration assez basique. Pour ajouter la définition du matériel rien de plus simple, il faut d'abord se connecter en cliquant sur "**Login**" en haut à droite, puis en cliquant au même endroit (affichant maintenant "**Logged in as Demo**") sélectionnez "**Machines**" dans le menu. Cliquez ensuite sur "***NEW MACHINE***" et renseignez les champs avec les valeurs suivantes :

Name	Backend	Settings
64bit	qemu	QEMUCPU=qemu64
		VIRTIO_CONSOLE=1
		WORKER_CLASS=qemu_x86_64

Cliquez ensuite sur l'icône représentant une disquette pour sauvegarder.

Explication rapide des variables :

- **QEMUCPU** positionne le type de serveur (correspond à l'option "-cpu" de qemu).
- **VIRTIO_CONSOLE** permet d'activer une console série virtuelle dans qemu, cela permet d'accélérer certains traitements (en lignes de commande principalement) plutôt que de d'utiliser tout le temps les *needles*. Activé par défaut en x86_64.
- **WORKER_CLASS** permet de faire le lien entre la "**Machine**" et les workers openQA (configurés lors de l'installation).

Type d'OS

Le type d'OS définit quel système d'exploitation doit s'exécuter. Tout naturellement j'ai choisi de baser le test sur l'OS openSUSE. Comme pour l'architecture matérielle, il faut sélectionner le menu "**Medium types**" puis "***NEW MEDIUM***" et renseigner les champs avec les valeurs suivantes :

Distri	Version	Flavor	Arch	Settings
opensuse	42.3	DVD	x86_64	DVD=1
				ISO_MAXSIZE=4700372992

Explication rapide des variables :
- **DVD** indique qu'on utilise directement le DVD pour l'installation (par opposition à une installation réseau).
- **ISO_MAXSIZE** définit la taille maximale de l'image DVD, pas indispensable ici mais cela fait généralement partie de la configuration standard. C'est une option qui sert surtout à vérifier qu'une image de DVD openSUSE ne dépasse pas la taille physique du DVD.

Définition des tests

La définition des tests permet de définir (ça tombe bien !) les tests que l'on veut exécuter. Sélectionnez le menu "**Test suites**" puis "***NEW TEST SUITE***" et renseignez les champs avec les valeurs suivantes :

Name	Settings
create_hdd_xfce	DESKTOP=xfce FILESYSTEM=xfs INSTALLONLY=1 PUBLISH_HDD_1=%DISTRI%-%VERSION%-%ARCH%-xfce@%MACHINE%.qcow2
test_scummvm_v1	BOOT_HDD_IMAGE=1 DESKTOP=xfce HDD_1=%DISTRI%-%VERSION%-%ARCH%-xfce@%MACHINE%.qcow2 SCUMMVM=1 BASS_SKIP_INTRO=1 START_AFTER_TEST=create_hdd_xfce
test_scummvm_v2	ALT_SCUMMVM_INSTALL=1 BOOT_HDD_IMAGE=1 DESKTOP=xfce HDD_1=%DISTRI%-%VERSION%-%ARCH%-xfce@%MACHINE%.qcow2 SCUMMVM=1 BASS_SKIP_INTRO=1 START_AFTER_TEST=create_hdd_xfce
test_scummvm_v3	BOOT_HDD_IMAGE=1 DESKTOP=xfce HDD_1=%DISTRI%-%VERSION%-%ARCH%-xfce@%MACHINE%.qcow2 SCUMMVM=1 SCUMMVM_FULLSCREEN=1 BASS_SKIP_INTRO=1 START_AFTER_TEST=create_hdd_xfce

Explication rapide des variables pour le test *create_hdd_xfce* :
- **DESKTOP** indique le type d'environnement graphique, ici Xfce.
- **FILESYSTEM** indique le type de filesystem à utiliser pour l'installation.
- **INSTALLONLY** indique de ne faire que la partie installation et donc de ne pas exécuter les tests additionnels.
- **PUBLISH_HDD_1** indique à la fois que ce test doit générer une image qcow2 à la fin ainsi que le nom de cette image.

Explication rapide des variables pour les tests *test_scummvm_v** :
- **BOOT_HDD_IMAGE** indique que l'on doit booter la VM à partir d'une image (celle générée via PUBLISH_HDD_1) plutôt que depuis le DVD.
- **DESKTOP** comme précédemment indique le type d'environnement graphique.
- **HDD_1** indique l'image à utiliser pour le premier disque, et donc l'image sur laquelle on doit booter.
- **SCUMMVM** indique que l'on va exécuter les tests **ScummVM**.
- **ALT_SCUMMVM_INSTALL** indique que l'on souhaite utiliser une méthode d'installation alternative pour **ScummVM**.
- **SCUMMVM_FULLSCREEN** indique que… je vous laisse deviner ! On reviendra sur cette option plus tard.
- **BASS_SKIP_INTRO** permet de contourner l'introduction du jeu, cela permet de gagner pas mal de temps lors des phases de test, en enlevant cette option l'introduction complète du jeu sera exécutée.
- **START_AFTER_TEST** indique que l'on doit exécuter le test ***après*** celui passé en paramètre, cela permet d'ordonnancer les tests (ici on ne peut rien exécuter tant que l'on n'a pas installé un OS).

Liens tests/OS/serveur

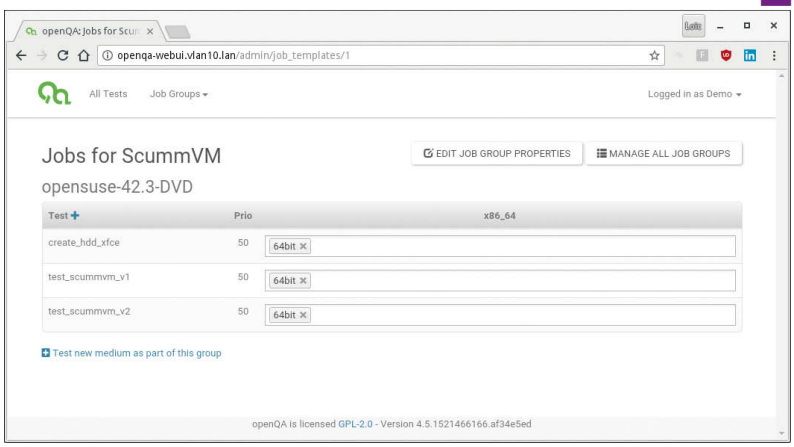
Les liens tests/OS/serveur permettent de définir sur quel OS et sur quelle architecture l'on souhaite exécuter les tests. Cela peut sembler un peu "lourd" dans le cadre de notre exemple avec un seul OS et une seule architecture, mais dans le cas des tests sur un OS comme openSUSE cela s'avère très pratique. Il suffit de renseigner les champs suivants en cliquant sur "+ ***Test new medium as part of this group***" :

Medium	Test Suite	Machine
opensuse-42.3-DVD-x86_64	create-hdd_xfce	64bit
opensuse-42.3-DVD-x86_64	test_scummvm_v1	64bit
opensuse-42.3-DVD-x86_64	test_scummvm_v2	64bit

Vous verrez alors apparaître les liens sous la forme suivante : 1
Dans les cas où il y a plusieurs architectures, des colonnes supplémentaires avec le type de matériel apparaissent.
Ouf ! Enfin la configuration est terminée, on va pouvoir passer à l'étape suivante.

Suite et fin de notre dossier openQA dans Programmez! 219

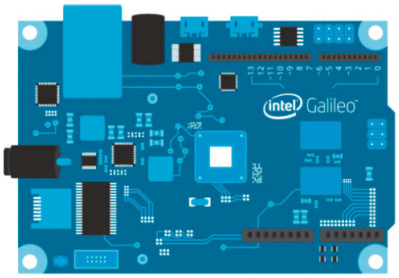
1



Pierre FICHEUX
pierre.ficheux@smile.fr
Directeur technique Smile ECS
Innovation @ Smile

Introduction à Yocto

Partie 2



Dans ce deuxième article nous allons nous attacher à l'ajout puis à la création de « recettes » permettant d'étendre les possibilités de l'image produite. La lecture du premier article est bien entendu nécessaire à la compréhension de ce deuxième volet.

Étendre l'image produite

Nous avions vu que la méthode de production de l'image était la même quelle que soit la cible choisie (dans notre cas QEMU/x86 et Raspberry Pi 3). Nous avions choisi de construire l'image la plus simple, soit « core-image-minimal », par les commandes suivantes dans le cas de la Pi 3.

```
$ cd <Yocto-src-path>
$ source oe-init-build-env rpi3-build
$ bitbake core-image-minimal
```

Le test de l'image dépend de la cible choisie puisqu'un test QEMU/x86 se résume à l'appel de l'émulateur (soit **runqemu qemux86**) alors qu'un test sur cible réelle nécessite de copier l'image produite sur une mémoire flash (ou un disque dur). Dans le cas de la Raspberry Pi 3, nous pouvons créer une Micro-SD par les commandes suivantes.

```
$ umount /dev/mmcblk0p*
$ sudo dd if=tmp/deploy/image/raspberrypi3/core-image-minimal-raspberrypi3.rpi-sdimg of=/dev/mmcblk0
```

L'image actuelle est réduite au strict minimum car elle n'inclut pas les modules du noyau Linux et ne dispose pas de système de gestion de paquets. De ce fait la taille de l'image produite est très faible (56 Mo pour l'image complète).

```
$ ls -lLh tmp/deploy/images/raspberrypi3/core-image-minimal-raspberrypi3.rpi-sdimg
-rw-r--r-- 1 pierre pierre 56M avril 11 15:25 tmp/deploy/images/raspberrypi3/core-image-minimal-raspberrypi3.rpi-sdimg
```

Il est donc judicieux d'étendre l'image en ajoutant d'autres paquets ou fonctionnalités. Dans un premier temps il est tout à fait possible de réaliser cela en modifiant le fichier **conf/local.conf**. A titre d'exemple nous pouvons ajouter un serveur SSH nommé *Dropbear* en ajoutant la ligne suivante au fichier **conf/local.conf** . Cette ligne indique d'ajouter à l'image finale le résultat de la construction de la recette correspondante. Cette recette est déjà fournie par Yocto dans le répertoire **meta/recipes-core/dropbear**.

```
IMAGE_INSTALL_append = " dropbear"
```

Nous avions vu que la construction de la recette passait par différentes étapes en partant du téléchargement des sources jusqu'à la création du paquet binaire. Notre image ne disposant pas de système de gestion de paquets, le nouveau composant est ajouté à l'image de manière statique (comme on le ferait avec Buildroot). Le paquet est cependant créé sur le poste de développement comme l'indique la commande suivante.

```
$ ls -l tmp/deploy/ipk/cortexa7hf-neon-vfpv4/dropbear*
-rw-r--r-- 2 pierre pierre 119004 avril 11 15:38 tmp/deploy/ipk/cortexa7hf-neon-vfpv4/dropbear_2017.75-r0_cortexa7hf-neon-vfpv4.ipk
-rw-r--r-- 2 pierre pierre 881666 avril 11 15:38 tmp/deploy/ipk/cortexa7hf-neon-vfpv4/dropbear-dbg_2017.75-r0_cortexa7hf-neon-vfpv4.ipk
-rw-r--r-- 2 pierre pierre  832 avril 11 15:38 tmp/deploy/ipk/cortexa7hf-neon-vfpv4/dropbear-dev_2017.75-r0_cortexa7hf-neon-vfpv4.ipk
```

Nous verrons plus tard pourquoi il existe ***trois*** paquets binaires pour une seule recette (soit des paquets **-dbg** et **-dev** en plus du paquet principal). La commande suivante indique quels sont les composants installés par le paquet binaire. Le format IPK étant proche du format DEB utilisé sur Debian/Ubuntu, le contenu du paquet est visible grâce à la commande **dpkg**.

```
$ dpkg -c tmp/deploy/ipk/cortexa7hf-neon-vfpv4/dropbear_2017.75-r0_cortexa7hf-neon-vfpv4.ipk
drwxrwxrwx root/root      0 2018-04-11 15:38 ./
drwxr-xr-x root/root      0 2018-04-11 15:38 ./etc/
drwxr-xr-x root/root      0 2018-04-11 15:38 ./etc/default/
drwxr-xr-x root/root      0 2018-04-11 15:38 ./etc/dropbear/
drwxr-xr-x root/root      0 2018-04-11 15:38 ./etc/init.d/
-rwxr-xr-x root/root   2168 2018-04-11 15:38 ./etc/init.d/dropbear
drwxr-xr-x root/root      0 2018-04-11 15:38 ./var/
drwxr-xr-x root/root      0 2018-04-11 15:38 ./usr/
drwxr-xr-x root/root      0 2018-04-11 15:38 ./usr/sbin/
-rwxr-xr-x root/root 224168 2018-04-11 15:38 ./usr/sbin/dropbearmulti
lrwxrwxrwx root/root      0 2018-04-11 15:38 ./usr/sbin/dropbearconvert -> ./dropbearmulti
lrwxrwxrwx root/root      0 2018-04-11 15:38 ./usr/sbin/dropbear -> ./dropbearmulti
lrwxrwxrwx root/root      0 2018-04-11 15:38 ./usr/sbin/dropbearkey -> ./dropbearmulti
drwxr-xr-x root/root      0 2018-04-11 15:38 ./usr/bin/
lrwxrwxrwx root/root      0 2018-04-11 15:38 ./usr/bin/dbclient -> /usr/sbin/dropbearmulti
```

Au démarrage de la nouvelle image, nous constatons l'initialisation et le démarrage de Dropbear.

```
Starting Dropbear SSH server: Generating key, this may take a while...
```

Outre l'ajout de recettes, nous avions vu dans le premier article qu'il était possible d'ajouter des « features » comme la prise en compte des paquets binaires. Cette fonctionnalité est indispensable si nous voulons tester le développement et l'ajout dynamique de recette. Pour ce faire nous ajoutons la ligne suivante au fichier **conf/local.conf** et nous construisons de nouveau l'image. Notons que nous avons commenté la ligne concernant le serveur Dropbear car nous l'ajouterons a posteriori en utilisant le système de gestion de paquets. Notons également qu'il est nécessaire de spécifier le format IPK car Yocto utilise par défaut le format RPM. Nous avons également augmenté la taille du root-filesystem de 50 Mo afin de pouvoir manipuler quelques paquets binaires.

```
EXTRA_IMAGE_FEATURES = "package-management"
PACKAGE_CLASSES = "package_ipk"
IMAGE_ROOTFS_EXTRA_SPACE = "50000"
#IMAGE_INSTALL_append = " dropbear"
```

Dans la suite de l'article nous partirons du principe que l'image utilise cette configuration. Au démarrage du système, celui-ci doit disposer de la commande **opkg**.

```
root@raspberrypi3:~# type opkg
opkg is a tracked alias for /usr/bin/opkg
```

Configurer le gestionnaire de paquets

Le gestionnaire de paquets utilise la base des paquets binaires présente sur le poste de développement dans le répertoire **tmp/deploy/ipk**. Comme nous l'avions vu dans le premier article, ce répertoire contient trois sous-répertoires correspondant aux trois types de paquets (indépendant de l'architecture, spécifique à l'architecture, spécifique à la cible).

```
$ ls -l tmp/deploy/ipk/
total 568
drwxr-xr-x 2 pierre pierre   4096 avril 11 16:33 all
drwxr-xr-x 2 pierre pierre 372736 avril 11 16:33 cortexa7hf-neon-vfpv4
drwxr-xr-x 2 pierre pierre 196608 avril 11 16:33 raspberrypi3
```

En premier lieu on doit construire l'index à partir de la liste des paquets.

```
$ bitbake package-index
```

On peut alors donner accès au répertoire par le protocole HTTP. Dans le cas de notre test, on peut utiliser le mini-serveur HTTP intégré à Python qui utilise le port 8000.

```
$ cd tmp/deploy/ipk/
$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
```

Du côté de la cible on doit modifier le fichier **/etc/opkg/opkg.conf** afin d'ajouter les caractéristiques du serveur. En supposant que l'adresse IP du PC de développement est 192.168.1.23, on ajoute les lignes suivantes au fichier.

```
# Local config
src/gz all http://192.168.1.23:8000/all
src/gz cortexa7hf-neon-vfpv4 http://192.168.1.23:8000/cortexa7hf-neon-vfpv4
src/gz raspberrypi3 http://192.168.1.23:8000/raspberrypi3
```

On peut alors mettre à jour l'index des paquets sur la cible.

```
root@raspberrypi3:~# opkg update
```

Cette commande a pour effet d'afficher les traces suivantes dans la fenêtre du serveur HTTP, l'adresse IP de la carte étant 192.168.1.16.

```
192.168.1.16 - - [11/Apr/2018 16:34:18] "GET /all/Packages.gz HTTP/1.1" 200 -
192.168.1.16 - - [11/Apr/2018 16:34:18] "GET /cortexa7hf-neon-vfpv4/Packages.gz HTTP/1.1" 200 -
192.168.1.16 - - [11/Apr/2018 16:34:18] "GET /raspberrypi3/Packages.gz HTTP/1.1" 200 -
```

On peut alors utiliser quelques options de la commande **opkg**. En premier lieu on peut obtenir la liste des paquets installés.

```
root@raspberrypi3:~# opkg list-installed
```

On peut ensuite afficher le contenu d'un paquet particulier ou bien des informations concernant ce paquet.

```
root@raspberrypi3:~# opkg files busybox
Package busybox (1.24.1-r0) is installed on root and has the following files:
/bin/sh
/bin/busybox.suid
/bin/busybox
/etc/busybox.links.nosuid
/etc/busybox.links.suid
/bin/busybox.nosuid
```

On peut également rechercher le paquet Dropbear et l'installer dynamiquement sur la cible. Le paquet est configuré lors de l'installation.

```
root@raspberrypi3:~# opkg list | grep dropbear
dropbear - 2017.75-r0 - A lightweight SSH and SCP implementation
dropbear-dbg - 2017.75-r0 - A lightweight SSH and SCP implementation - Debugging files
dropbear-dev - 2017.75-r0 - A lightweight SSH and SCP implementation - Development files

root@raspberrypi3:~# opkg install dropbear
Installing dropbear (2017.75) on root
Downloading http://192.168.1.23:8000/cortexa7hf-neon-vfpv4/dropbear_2017.75-r0_cortexa7hf-neon-vfpv4.ipk.
...
```

Créer des recettes

Nous pouvons maintenant nous attacher à créer quelques recettes simples que nous pourrons ajouter à notre environnement Yocto puis à notre cible. Une recette est un fichier **.bb** (pour BitBake) utilisant une syntaxe décrivant des métadonnées. Les directives de la recette sont exécutées par la commande **bitbake**. La notion de recette est utilisée quelle que soit la complexité du composant (un simple exemple « Hello World », l'outil BusyBox, le noyau Linux et même l'image du système).

```
$ cd meta
$ ls recipes-core/images
build-appliance-image       core-image-minimal-dev.bb
build-appliance-image_15.0.0.bb  core-image-minimal-initramfs.bb
core-image-base.bb      core-image-minimal-mtdutils.bb
core-image-minimal.bb       core-image-tiny-initramfs.bb
recipes-core/images/core-image-minimal.bb

$ ls recipes-core/busybox
busybox  busybox-1.24.1  busybox_1.24.1.bb  busybox.inc  files
```

La recette du noyau est fournie par la couche (layer) BSP externe à Yocto et disponible auprès du fabricant de la carte ou d'une communauté de développement (comme c'est le cas pour la Raspberry Pi).

```
$ cd ../meta-raspberrypi
$ ls recipes-kernel/linux
linux-raspberrypi_4.9.bb  linux-raspberrypi-dev.bb  linux-raspberrypi.inc
```

Comme nous pouvons le constater, une recette est toujours associée à un layer (**meta**, **meta-raspberrypi**, **meta-<nom-de-layer>**) dans un sous-répertoire de type **recipes-*/<nom-de-recette>** .

Les différents types de recettes

Nous pouvons classifier (schématiquement) les recettes en quelques types bien identifiés. Ces types sont déduits des outils utilisés pour la production des projets libres dans un environnement de type UNIX/Linux (même si ces outils existent également pour d'autres systèmes d'exploitation).

- Utilisation de l'outil ***Autotools*** [3];
- Utilisation de l'outil ***CMake*** [4];
- Plus rarement, utilisation de la commande **make** et d'un fichier **Makefile**.

Il n'est bien entendu pas envisageable de décrire les outils Autotools et CMake dans le cadre de cet article et nous renvoyons donc le lecteur aux entrées de bibliographie. Ces outils sont prévus pour permettre la compilation (croisée ou non) d'un projet sur (et à destination de) différents types de systèmes. L'utilisation d'un simple **Makefile** est un peu « démodée » car ce dernier ne prend pas automatiquement en compte les caractéristiques des systèmes comme le font Autotools et CMake.

Yocto permet de créer des recettes sans réellement se préoccuper du fonctionnement de ces outils en utilisant la notion de classes. Les principales classes disponibles sont localisées dans le répertoire **meta/classes** sous la forme de fichiers **.class**. A titre d'exemple, le fichier suivant correspond à la classe permettant de traiter les sources basées sur Autotools.

```
$ ls -l meta/classes/autotools.bbclass
-rw-rw-r-- 1 pierre pierre 8798 févr.  9 22:25 meta/classes/autotools.bbclass
```

Comme nous le verrons dans les exemples qui suivent, on peut utiliser une classe en invoquant simplement la directive **inherit <nom-de-classe>** dans le fichier de recette.

Un exemple (très) simple

Avant de décrire le format d'une recette, nous allons utiliser l'outil **yocto-layer** fourni par Yocto afin de créer un layer de test **meta-example**. Outre la création du layer, **yocto-layer** permet d'ajouter automatiquement une recette de test au layer créé. Pour cela on utilise la commande suivante à partir du répertoire des sources de Yocto (ou même dans un répertoire quelconque).

```
$ cd <Yocto-src-path>
$ yocto-layer create example
Please enter the layer priority you'd like to use for the layer: [default: 6]
Would you like to have an example recipe created? (y/n) [default: n] y
Please enter the name you'd like to use for your example recipe: [default: example]
Would you like to have an example bbappend file created? (y/n) [default: n]

New layer created in meta-example.

Don't forget to add it to your BBLAYERS (for details see meta-example/README).
```

Une nouvelle recette **example** est créée dans l'arborescence. La recette est constituée du fichier **.bb** ainsi que d'un répertoire contenant les données (dans notre cas les sources de l'exemple).

```
$ tree meta-example/
meta-example/
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
├── recipes-example
│   └── example
│       ├── example-0.1
│       │   ├── example.patch
│       │   └── helloworld.c
│       └── example_0.1.bb
```

Comme l'indique le dernier message produit par **yocto-layer**, on doit ajouter le nouveau layer à l'environnement de compilation. Cette action est réalisée par la commande **bitbake-layers**.

```
$ cd rpi3-build
$ bitbake-layers add-layer ../meta-example
```

Cette commande ajoute simplement le chemin d'accès du layer au fichier **conf/bblayers.conf**. On peut alors construire la recette (i.e. produire le paquet binaire).

```
$ bitbake example
```

Les paquets sont disponibles dans le répertoire prévu.

```
$ ls -l tmp/deploy/ipk/cortexa7hf-neon-vfpv4/example*
-rw-r--r-- 2 pierre pierre 2176 avril 11 18:27 tmp/deploy/ipk/cortexa7hf-neon
-vfpv4/example_0.1-r0_cortexa7hf-neon-vfpv4.ipk
-rw-r--r-- 2 pierre pierre 3960 avril 11 18:27 tmp/deploy/ipk/cortexa7hf-neon-
vfpv4/example-dbg_0.1-r0_cortexa7hf-neon-vfpv4.ipk
-rw-r--r-- 2 pierre pierre  732 avril 11 18:27 tmp/deploy/ipk/cortexa7hf-neon-
vfpv4/example-dev_0.1-r0_cortexa7hf-neon-vfpv4.ipk
```

Le paquet « principal » contient l'exécutable **helloworld** issu de la compilation du fichier **helloworld.c**.

```
$ dpkg -c tmp/deploy/ipk/cortexa7hf-neon-vfpv4/example_0.1-r0_cortexa7hf-neon-
vfpv4.ipk
drwxrwxrwx root/root      0 2018-04-11 18:27 ./
drwxr-xr-x root/root      0 2018-04-11 18:27 ./usr/
drwxr-xr-x root/root      0 2018-04-11 18:27 ./usr/bin/
-rwxr-xr-x root/root   5552 2018-04-11 18:27 ./usr/bin/helloworld
```

Le paquet **-dbg** contient une version de l'exécutable « débogable » sur la cible (en utilisant **gdbserver**). Le paquet **-dev** est utile dans le cas des recettes de bibliothèques pour lesquelles il est nécessaire de disposer du fichier binaire (le **.so**) sur la cible, mais également dans l'environnement de développement croisé (SDK) produit par Yocto. Dans le cas d'un exécutable, ce paquet est bien entendu vide.

Avant d'installer le nouveau paquet sur la cible, on doit mettre à jour l'index sur le PC de développement.

```
$ bitbake package-index
```

On doit également effectuer la mise à jour (update) de la liste sur la cible.

```
root@raspberrypi3:~# opkg update
Downloading http://192.168.1.23:8000/all/Packages.gz.
Updated source 'all'.
Downloading http://192.168.1.23:8000/cortexa7hf-neon-vfpv4/Packages.gz.
Updated source 'cortexa7hf-neon-vfpv4'.
Downloading http://192.168.1.23:8000/raspberrypi3/Packages.gz.
Updated source 'raspberrypi3'.
```

On peut alors installer et tester le nouveau paquet.

```
root@raspberrypi3:~# opkg install example
Installing example (0.1) on root
Downloading http://192.168.1.23:8000/cortexa7hf-neon-vfpv4/example_0.1-r0_cortex
a7hf-neon-vfpv4.ipk.
Configuring example.

root@raspberrypi3:~# helloworld
Hello World!
```

Syntaxe du fichier de recette

Après avoir installé le paquet nous allons décrire rapidement le format du fichier de recette. Les premières lignes décrivent le composant ainsi que la catégorie. Ces informations seront utilisées par le gestionnaire de paquets.

```
SUMMARY = "Simple helloworld application"
SECTION = "examples"
```

On déclare ensuite la licence des sources que l'on authentifie grâce à un checksum.

```
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
```

La variable SRC_URI doit contenir la liste de ***tous*** les composants nécessaires à la construction du paquet. Dans notre cas la liste se limite à un fichier source local (d'où l'opérateur **file**://) présent dans le sous-répertoire example_0.1.

```
SRC_URI = "file://helloworld.c"
```

Les lignes suivantes décrivent la compilation du fichier source et l'installation de l'exécutable. Le fichier Makefile étant absent on doit décrire précisément comment réaliser ces opérations en invoquant les commandes nécessaires.

```
S = "${WORKDIR}"

do_compile() {
        ${CC} ${LDFLAGS} helloworld.c -o helloworld
}

do_install() {
        install -d ${D}${bindir}
        install -m 0755 helloworld ${D}${bindir}
}
```

Recettes Autotools et CMake

Dans la réalité la plupart des projets utilisent Autotools, CMake ou plus rarement un simple fichier Makefile. Dans cette section nous allons voir comment écrire une recette en utilisant les classes Autotools et CMake, ce qui facilite grandement la tâche. Considérons un exemple « Hello World » proche du précédent mais basé sur Autotools. Le contenu du fichier de recette est décrit ci-dessous.

```
DESCRIPTION = "Helloworld software (autotools)"

LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=8ca43cbc842c2336e835926c2166c28b"

SRC_URI = "http://pficheux.free.fr/pub/tmp/mypack-auto-1.0.tar.gz"

inherit autotools

SRC_URI[md5sum] = "b282082e4e5cc8634b7c6caa822ce440"
```

La première différence avec le premier exemple est l'obtention des sources auprès d'un serveur externe (ce qui est le cas le plus fréquent). A partir du moment ou une archive externe est utilisée, on doit déclarer le ***checksum*** du fichier comme décrit dans la dernière ligne de la recette. Le traitement des sources est entièrement pris en charge par la classe et il suffit donc d'utiliser la directive inherit. Le cas d'une recette basée sur CMake est identique en utilisant la classe correspondante.

```
DESCRIPTION = "Helloworld software (CMake)"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=8ca43cbc842c2336e835926c2166c28b"
PR = "r0"

SRC_URI = "http://pficheux.free.fr/pub/tmp/mypack-cmake-1.0.tar.gz"

inherit cmake

SRC_URI[md5sum] = "70e89c6e3bff196b4634aeb5870ddb61"
```

Conclusion

Ce deuxième article nous a permis d'effleurer les techniques de création des recettes Yocto ainsi que leur intégration dans une image. Le lecteur désirant aller plus loin pourra consulter l'ouvrage de l'auteur en [5] ou bien d'autres ouvrages dédiés à Yocto en [6].

Bibliographie

[1] Projet Yocto https://www.yoctoproject.org/
[2] Manuel de référence Yocto http://www.yoctoproject.org/docs/2.4.1/ref-manual/ref-manual.html
[3] Autotools tutorial (LRDE) https://www.lrde.epita.fr/~adl/autotools.html
[4] Tutoriel CMake
http://sirien.metz.supelec.fr/depot/SIR/TutorielCMake/index.html
[5] « Linux embarqué, mise en place et développement » (P Ficheux) https://www.eyrolles.com/Informatique/Livre/linux-embarque-9782212674841
[6] Quelques ouvrages dédiés à Yocto
https://www.yoctoproject.org/learn/books

Julien Demangeon
Consultant / Développeur Full-Stack
Marmelab - marmelab.com

Crystal, le langage qu'il vous faut ? Partie 1

Le langage Crystal a été publié pour la première fois en 2014 et est encore activement développé. Il fait aujourd'hui partie des langages compilés à considérer pour démarrer un nouveau projet. Pourquoi ce langage plutôt qu'un autre ? À quel type de projet est-il le plus adapté ? Nous allons tenter de répondre à ces questions.

Une syntaxe tirée de Ruby

La syntaxe utilisée par Crystal est très proche (voir semblable) à celle du langage Ruby, et pour cause : ses créateurs apprécient énormément ce dernier. Pour ceux qui ne connaissent pas, Ruby est un langage dynamique, interprété et entièrement orienté-objet, développé il y a plus de 20 ans. Il tire principalement sa popularité de son framework Web phare, Ruby on Rails (aka RoR).
En dehors de cette affinité syntaxique, Crystal utilise aussi de nombreux concepts de Ruby tels que les Blocks, les Symbols et le paradigme Tout objet.

Blocks

Les blocks permettent de capturer une portion de code et son contexte d'exécution afin d'être exécutés ultérieurement. On les compare souvent à ce que l'on appelle closures ou fonctions anonymes dans d'autres langages.

```
(1...5).each do |idx|
  idx # => 1, 2, 3, 4
end

# (1...5) === Range.new(1, 5, exclusive: true)
```

Dans cet exemple très simple, le block défini entre |idx| et end est appelé pour chaque nombre de l'intervalle [1-5[(5 est exclu grâce au "..." plutôt que "..").
L'expression "(1...5)" est en réalité transformée en un objet de type Range qui possède une méthode each acceptant un objet de type Block. À l'intérieur de cette méthode each, le block passé en argument est exécuté à l'aide de l'instruction yield. Ce mode de fonctionnement se rapproche beaucoup des générateurs que l'on retrouve dans de nombreux autres langages tels que Python ou ES6.

```
def twice
  yield "foo"
  yield "bar"
end

twice do |name|
  puts "Hello #{name}!"
end

# => "Hello foo!"
# => "Hello bar!"
```

Dans cet exemple, le bloc "do / end" situé après twice est exécuté 2 fois grâce à l'instruction "yield" de la fonction "twice". Bien entendu, il n'est pas obligatoire de passer des arguments à cette instruction.

Symbols

Le Symbol est un concept très intéressant que l'on retrouve aussi dans Ruby. Un Symbol est en réalité une constante qui est identifiée par son nom précédé de ":", comme dans l'exemple ci-dessous.

```
:hello === :hello # true
:hello === "hello" # false
```

Crystal se charge en interne de maintenir un registre de l'ensemble des Symbols utilisés (représentés par un Int32).
On retrouve également ce principe de Symbol en Elixir et en Erlang à travers un nom différent, les Atoms.

Principe du tout objet

En Crystal, tout comme en Ruby, TOUT est objet. D'ailleurs, depuis le début de la lecture de cet article, vous avez rencontré au moins 4 types d'objets différents (Blocks, Symbols, Ranges et Ints).
Tout comme en Ruby, tous les objets étendent l'objet primitif nommé "Object". C'est pourquoi, la plupart du temps, lorsque vous développez en Crystal, vous développez des Classes.

```
module Curses
  class Window
    def new
      # ...
    end
  end
end

Curses::Window.new
```

Cependant, Crystal peut aussi être utilisé de manière "fonctionnelle". Étant donné que chaque "module" nécessite une classe, la solution est de rendre le module "self-extended". Exemple.

```
module Curses
  extend self

  def new
    # ...
  end
end
```

Il est maintenant possible d'appeler la fonction "new" directement sans avoir à passer par une classe.

```
include Curses
new()
```

Typage Statique et Inférence

L'un des principaux avantages de Crystal par rapport à d'autres langages compilés est son système d'inférence de type. Ce processus consiste à donner la responsabilité d'attribution des types à Crystal lors de la compilation.
Par ailleurs, Crystal possède un système appelé "Union Type" qui permet de donner plusieurs types à une seul et même variable.

```
if 1 + 2 == 3
 a = 1
else
 a = "hello"
end

a # : Int32 | String
```

Dans l'exemple ci-dessus, le type est automatiquement attribué à la compilation à partir de l'AST (Abstract Syntax Tree). L'AST est une représentation virtuelle de "l'arbre" de code tel qu'il a été "interprété" par le compilateur, d'où son nom. En Crystal, il est aussi tout à fait possible d'attribuer manuellement un type.

```
a : (Int32 | String) = 1
```

L'inférence de type fonctionne également pour de nombreux autres types de données tels que les Arrays, Tuples, NamedTuples, Hashes, etc.

```
["hello", 'x'] # Array(String | Char)
{1, "hello"} # Tuple(String, Char)
{x: "Crystal", y: 2017} # NamedTuple(x: String, y: Int32)
{1 => 2, 'a' => 3} # Hash(Int32 | Char, Int32)
```

Des structures de données vides peuvent aussi être déclarées à l'aide de certains raccourcis.

```
Tuple(Int32, Char)
NamedTuple(x: Int32, y: String)
[] of Int32 # = Array(Int32).new
{} of Int32 => Int32 # = Hash(Int32, Int32).new
```

Null Reference Checks

Dans votre parcours, vous avez certainement déjà rencontré ou entendu parler des "NullPointerException" ou bien de la fameuse "One Billion Dollar Mistake". Ce concept d'erreur, bien connu dans le développement, vient du fait qu'il est très facile d'introduire des appels à une référence dont la valeur est "null".
En Crystal, ce type d'exception ne peut pas survenir. En effet, lors de la compilation et de l'attribution des types par inférence, une vérification est effectuée pour empêcher cela. Voici un exemple minimaliste permettant d'illustrer ce principe.

```
if rand(42) > 0
 hello = "hello world"
end

puts hello.upcase
```

Le code ci-dessus sera rejeté par Crystal lors de la compilation. L'exception suivante sera levée par le compilateur :

```
$ crystal hello_world.cr
Error in hello_world.cr:5: undefined method 'upcase' for Nil (compile-time type is
 (String | Nil))

puts hello.upcase
     ^~~~~~
```

En effet, et comme le message de l'exception l'indique parfaitement bien, la variable "hello" peut être de type String ou Nil (équivalent de "Null" en Java). Or, il n'existe pas de méthode "upcase" sur le type Nil. La méthode "upcase" ne peut donc pas être appelée sur "hello".

Metaprogramming et Macros

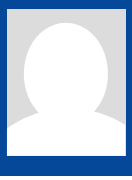
Comme il a été possible de le voir depuis le début de cet article, l'introspection et l'inférence de types à travers un AST est une fonctionnalité centrale du langage. En réalité, elle offre bien plus encore.
Effectivement, Crystal possède un système de "macros". Une "macro" est un processus par lequel il est possible de modifier, d'étendre ou d'ajouter du code au langage de manière dynamique (ici à la compilation). Ce processus de génération de code est plus généralement appelé "Metaprogramming".
Cette faculté étonnante permet au langage Crystal d'être lui-même développé en grande partie en... Crystal ! D'autres langages tels que Elixir sont également développés de cette manière.
Jugez-en par vous-même par un petit exemple tiré de la documentation.

```
# Macro

macro define_method(name, content)
 def {{name}}
  {{content}}
 end
end

# Appel à la génération

define_method foo, 1

# Code généré

def foo
 1
end

# Utilisation

foo # => 1
```

Les macros sont souvent utilisées dans des librairies qui permettent l'usage d'une syntaxe spécifique par le développeur.
Par exemple, la célèbre librairie d'abstraction de base de données Crecto (https://github.com/Crecto/crecto) utilise des macros afin de décorer ses modèles de données (ici avec la relation "belongs_to").

```
class Post < Crecto::Model
 schema "posts" do
  belongs_to :user, User
 end
end
```

La suite dans Programmez! 219

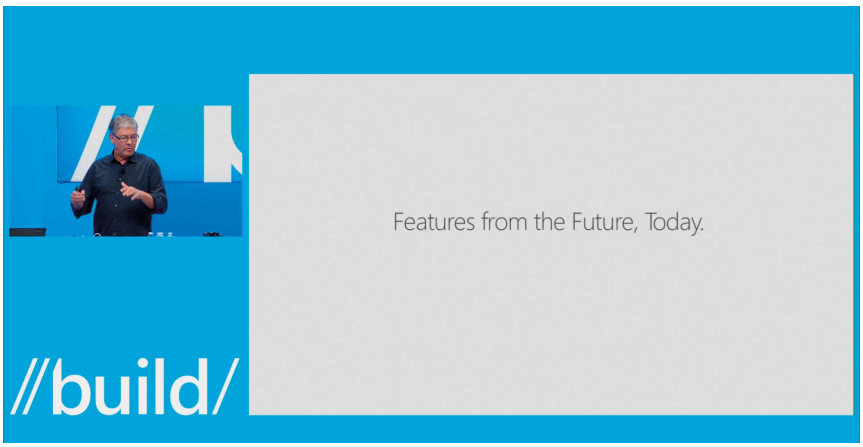


Sylvain Pontoreau
Premier Field Engineer
sylvain.pontoreau@microsoft.com
https://sylvain.pontoreau.com

Programmer avec TypeScript

Partie 1

Beaucoup de langages de programmation ont vu le jour ces dernières années. Parmi eux il y a TypeScript, un langage multi-paradigme dont le slogan affirme fièrement : « JavaScript that scales ». Dans ce dossier en trois parties, nous allons revenir ensemble sur ce langage qui prend de plus en plus d'ampleur dans la communauté Web.



Back to 2014, lors de la ngEurope à Paris, l'équipe d'Angular annonce qu'elle travaille sur une nouvelle version du Framework. Pour cette refonte, les capacités d'ECMAScript 6 ne suffiront pas. L'équipe Google souhaite alors créer AltScript, un « *superset* » de JavaScript qui disposerait des capacités nécessaires au bon développement du projet. TypeScript avait aussi été étudié par l'équipe de Google, mais il avait alors considéré que le langage de Microsoft n'allait pas assez loin.

Finalement, AltScript ne verra jamais le jour. L'équipe de TypeScript proposera simplement d'enrichir le langage avec les capacités nécessaires au développement d'Angular. Pour la première fois depuis des années, Microsoft et Google vont travailler de concert sur un duo technologique : Angular & TypeScript. La vitrine que propose Angular va propulser le langage sur le devant de la scène, à tel point qu'à la sortie de la nouvelle version du Framework, le taux d'installation de TypeScript va être multiplié par trois (passant pour la première fois la barre symbolique d'un million d'installations hebdomadaires). Une popularité soudaine et inattendue pour ce langage qui a mis du temps à convaincre et trouver ses marques. Un palier est à présent franchi pour TypeScript et la sortie de la version 2 du langage va marquer un tournant. Petit à petit, TypeScript devient un « *first-class citizen* », de plus en plus de bibliothèques JavaScript le supportent officiellement et de nombreuses entreprises l'utilisent pour construire des applications d'envergure. Dans cette première partie, nous allons revenir sur l'historique du langage, sa philosophie, les outils pour développer avec, et les bases qu'il offre. Bienvenue dans le monde de TypeScript, un langage passionnant où les développeurs pratiquent le JavaScript du futur, mais dans le présent !

Historique et philosophie

La première version de TypeScript est sortie officiellement en 2012. Cette première version n'a pas vraiment attiré les développeurs de la communauté JavaScript qui se sont demandés quel était réellement l'intérêt d'utiliser ce langage. À l'époque, CoffeeScript existait déjà et proposait aussi de transformer un code intermédiaire en JavaScript. Cependant, la différence entre TypeScript et CoffeeScript est que ce dernier définit une syntaxe qui lui est propre. TypeScript se décrit comme un « *superset* » de JavaScript. En conséquence une ligne de code écrite en JavaScript est de facto du code TypeScript. Cette philosophie ne fait pas de TypeScript un remplaçant potentiel de JavaScript (contrairement à Dart, qui au départ a été poussé par Google dans ce sens), mais bien un surensemble typé de JavaScript. Certains concepts présents dès le lancement de TypeScript ont aussi fait débat dans la communauté JavaScript. On peut prendre l'exemple des interfaces, un concept que TypeScript met en œuvre, mais qui en réalité n'existe pas en JavaScript. D'ailleurs, je me rappelle bien qu'à l'époque je faisais aussi partie des réfractaires et je n'étais pas vraiment séduit à l'idée d'utiliser des concepts qui n'étaient pas présent nativement en JavaScript.

D'un autre côté, il y a une communauté qui a été particulièrement réceptive à la sortie de TypeScript, celle de C#. La raison de cet intérêt ? Simplement parce que TypeScript, de premier abord, ressemble fortement à C# et Java. Les développeurs pratiquant la POO traditionnelle depuis des années retrouvèrent donc certains concepts qui n'étaient pas encore normalisés par ECMA (ES6 étant sortie en 2015). On peut prendre l'exemple des classes notamment. Outre ces similitudes, le fait qu'Anders Hejlsberg soit aux commandes du projet a aussi rassuré sur la volonté de Microsoft de faire de TypeScript un projet à long terme. Pour ceux d'entre vous qui ne connaissent pas ce monsieur, il est surtout connu pour avoir travaillé sur Delphi et il est ni plus ni moins que le créateur de C#. C'est une des rares personnes chez Microsoft ayant le titre de « *Technical Fellow* », la plus haute distinction que l'on puisse obtenir dans l'entreprise en tant qu'ingénieur. Cela donna rapidement au langage une certaine crédibilité auprès des développeurs C#. Anders a beaucoup communiqué sur TypeScript lors d'événements officiels pour y rappeler les fondements et principes du langage. Il y a d'ailleurs un élément qu'il rappelle régulièrement et qui explique en partie que de plus en plus de développeurs utilisent TypeScript, le langage permet d'utiliser les fonctionnalités qui seront disponibles à l'avenir avec ECMAScript. C'est un des points qui fait la force de TypeScript, car en réalité c'est le JavaScript de demain que l'on peut utiliser aujourd'hui ! L'autre gros atout est la capacité de vérifier les types, limitant ainsi les potentielles erreurs et offrant des capacités de refactoring particulièrement efficaces.

Par où commencer ?

En voilà une bonne question ! Déjà il est important de comprendre un point, bien que TypeScript ressemble fortement à C#/Java, c'est en réalité du JavaScript qui va être exécuté. Il est donc primordial d'avoir de bonnes bases avec ce langage.

Si vous débutez en JavaScript, je vous conseille très fortement de jeter un coup d'œil au langage après la lecture de cet article, car le niveau des prochaines parties du dossier va fortement augmenter. Je conseille souvent au débutant de jeter un coup d'œil sur le Mozilla Developer Network (https://developer.mozilla.org), car je trouve cette documentation très simple et claire pour quelqu'un qui n'a pas beaucoup d'expérience en JavaScript.

Maintenant que vous êtes prévenu, passons à la suite. Assez tôt dans la vie du projet, l'équipe de TypeScript a eu la bonne idée de proposer le compilateur sous la forme d'un package Node.js. Cela permet de l'utiliser sur n'importe quel OS. En fonction de votre environnement de développement, les étapes d'installation peuvent être différentes. Si vous avez Visual Studio sur votre ordinateur, il y a de fortes chances que vous ayez déjà tout ce qu'il faut pour démarrer avec TypeScript. Cependant, si vous souhaitez utiliser un éditeur de texte comme Atom, VS Code ou encore Sublime Text, vous aurez besoin d'installer TypeScript globalement avec NPM :

```
$ npm install -g typescript
```

Une fois que TypeScript est installé, le compilateur est disponible sous la forme d'une CLI (« *Command-line interface* »). La commande permettant d'interagir avec le compilateur est : « *tsc* ». Elle peut être appelée directement pour compiler le contenu d'un répertoire, mais elle dispose aussi de nombreux arguments comme par exemple :

- --init : qui permet d'initialiser la configuration du compilateur pour démarrer un projet.
- --watch : qui lance le compilateur en mode observation et permet de le déclencher automatiquement si un fichier a été modifié ou créé.
- --locale : qui permet de changer la langue des messages d'erreur.

Une fois que vous avez installé TypeScript, il ne vous reste plus qu'à choisir votre outil de développement. À titre personnel, j'utilise uniquement Visual Studio Code lorsque je développe en TypeScript. Il est lui-même écrit avec le langage et dispose d'un support absolument excellent (Refactoring, Auto-completion, Correction automatique, debugging …). Tout ce qui est nécessaire au développement avec TypeScript est embarqué par défaut et en plus de cela il existe des dizaines d'extensions autour du langage. C'est surement le meilleur choix disponible pour les développeurs souhaitant se lancer dans TypeScript.

Avant d'écrire vos premières lignes de code, il est important d'expliquer rapidement comment utiliser des bibliothèques développées en JavaScript. Comme vous vous en doutez surement, elles sont majoritairement écrites avec JavaScript. Il est donc primordial de disposer d'un fichier de définition pour les utiliser de manière typé (l'extension de ces fichiers est : « *.d.ts* »). Depuis la version 2.0 du langage, la gestion des fichiers de définition a changé et aujourd'hui il faut utiliser NPM pour les installer dans votre projet. Il existe une organisation NPM où sont regroupées toutes les définitions compatibles avec la version 2.0 de TypeScript. Cette organisation s'appelle @types. Voici un exemple d'installation pour la définition de jQuery :

```
$ npm install --save-dev @types/jquery
```

Il faut aussi savoir que de plus en plus de bibliothèques supportent TypeScript. Certaines d'entre elles embarquent la définition directement dans le package. Le mieux reste de consulter la documentation de la bibliothèque pour savoir s'il est nécessaire d'installer le fichier de définition ou non.

Types et variables

Avec TypeScript vous êtes venu pour ça et il est donc logique de démarrer avec la déclaration des types. Première chose à savoir, le langage étant aligné sur ECMAScript, les types primitifs disponibles sont les mêmes que dans le standard (exemple : boolean, number, string…). La déclaration de variables est cependant différente de C#/Java ou encore ECMAScript 5. Voici une comparaison syntaxique par rapport à C#, ECMAScript 5 et TypeScript :

```
//C#
string name = "Bill";

//JavaScript (ES 5)
var name = "Bill";

//TypeScript
var name: string = "Bill"
```

Avec TypeScript, pour déclarer une variable, le type ne se suffit pas à lui-même. Il faut utiliser un mot clé qui va définir le « *scope* » (portée dans la langue de Molière) et le type de votre variable. Pour information le compilateur de TypeScript permet aussi d'utiliser l'inférence de type, il est donc possible de déclarer une variable sans préciser le type et cela ne posera aucun problème. Idem pour Visual Studio Code : 1

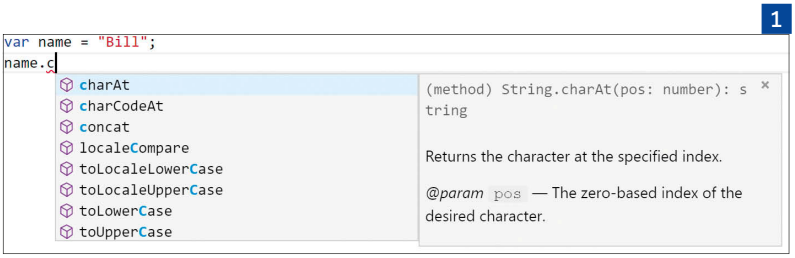
Le « *scope* » est un héritage direct de JavaScript et il est possible d'utiliser 3 valeurs : var, let et const. En ECMAScript 5, seul var est disponible, on retrouve donc cette portée partout et la règle est claire : lors de la déclaration de la variable avec var, si elle est à l'intérieur d'une fonction, sa portée est locale, sinon elle est globale. Cette portée, c'est le « *legacy* » de JavaScript. Elle est parfois difficile à comprendre pour les débutants, en plus d'être une source d'erreurs récurrente dans les projets.

Dans les nouvelles versions d'ECMAScript, une nouvelle portée est disponible : let. Ici les choses deviennent plus simples puisque la portée se retrouve cloisonnée au bloc dans lequel elle est déclarée. Il n'y a donc pas de notion locale ou globale. La bonne pratique avec TypeScript est donc d'utiliser let à la place de var afin de limiter les possibles erreurs :

```
let name: string = "Bill"
```

Et const dans l'histoire ? En réalité via ce mot clé les règles de portée de la variable seront les mêmes que let. La seule différence c'est que const inclut que la variable est « immutable ». Pour faire simple,

1



une fois que la variable a été créée, elle ne peut pas être altérée, on ne peut pas modifier sa référence :

```
const name: string = "Bill";
name = "Satya"; //Le compilateur déclenchera une erreur
```

La notion de portée avec JavaScript est un peu perturbante pour les débutants, c'est pourquoi je donne toujours comme bonnes pratiques de :

- déclarer en premier une variable avec const,
- changer const en let si vous devez modifier la variable,
- ne jamais utiliser var (sauf exception pour un besoin réel).

Les classes

TypeScript est aligné sur la prochaine version d'ECMAScript, les classes sont donc disponibles dans le langage depuis le début. Rien de révolutionnaire ici, on parle du concept de classe que l'on connaît en POO depuis des années. Si vous êtes un développeur C# ou Java vous ne serez pas perdu, car il y a de nombreuses similitudes avec TypeScript. Voici un exemple de déclaration d'une classe :

```
class Person {
  name: string;

  constructor(name: string) {
    this.name = name;
  }

  getHello(): string {
    //TypeScript supporte les chaînes interpolées :)
    return `Hello ${ this.name }!`;
  }
}

//Instanciation
const person: Person = new Person("Bill");
const hello = person.getHello();
console.log(hello);//Hello Bill!
```

Afin de vous donner un niveau de comparaison, voici la même classe écrite en ECMAScript 5, C# et Java : 2

Mis à part la compacité du code, on note pas mal de petites différences dans la syntaxe et le nommage. Plutôt que de passer en revue toutes ces différences, je vais plutôt m'attarder aux spécificités. Premièrement, TypeScript utilise le mot clé « *constructor* » pour en définir un, et non le nom de la classe comme pour C# ou Java. Deuxièmement, dans l'exemple je n'ai pas précisé l'accès aux membres d'une classe, car tout est public par défaut. Il est tout à fait possible de changer cela, TypeScript supportant trois niveaux d'accès aux membres :

- public : accessible à l'extérieur de la classe.
- private : accessible uniquement dans la classe.
- protected : idem que private, mais accessible depuis les classes dérivées.

Il est aussi possible de marquer un membre « *readonly* ». Il ne pourra alors être initialisé qu'à la déclaration ou dans le constructeur :

```
class Person {
  readonly firstName: string;
  readonly lastName: string = "GATES";//OK

  constructor(firstName: string) {
    this.firstName = firstName;//OK
  }
}

const person: Person = new Person("Bill");
person.lastName = "NADELLA";//Erreur
```

Pour finir sur les accès, il n'est pas possible de le préciser sur une classe. En TypeScript pour exposer une classe à l'extérieur il convient d'utiliser le mot clé « *export* ». Cette particularité vient de la notion de module défini dans ECMAScript. Nous reviendrons en détails sur ce point dans une autre partie du dossier, car il y a vraiment beaucoup à dire sur le sujet.

Voyons rapidement une autre possibilité intéressante. Les membres de la classe peuvent être marqués avec le mot clé static. Ils sont alors utilisables sans avoir besoin d'une instance de la classe :

```
class Person {
  static fullName: string = "Bill GATES";

  static sayHello() {
    console.log(`Hello ${Person.fullName}!`);
  }
}
```

2

```
//ECMAScript 5
var Person = function (name) {
    this.name = name;
};

Person.prototype.getHello = function () {
    return "Hello " + this.name + "!";
};

//Instanciation
var person = new Person("Bill");
var hello = person.getHello();
console.log(hello); //Hello Bill!
```

```
//C#
public class Person
{
    public string Name { get; set; }

    public string getHello()
    {
        return $"Hello { this.Name }!";
    }
}

//Instanciation
Person person = new Person();
person.Name = "Bill";
string hello = person.getHello();
Console.WriteLine(hello);//Hello Bill!
```

```
//Java
public class Person {
    private String name;

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getHello() {
        return String.format("Hello %s!", this.name);
    }
}

//Instanciation
Person person = new Person();
person.setName("Bill");
String hello = person.getHello();
System.out.println(hello);//Hello Bill!
```

```
Person.sayHello();//Hello Bill GATES!
```

Héritage

Pour finir cette première partie, attardons-nous sur l'héritage avec TypeScript. Une classe peut hériter d'une autre classe via le mot clé « *extends* ». Elle récupère alors les caractéristiques de la classe parente (y compris les membres déclarés protected) :

```
class Person {
  name: string = "Bill GATES";

  sayHello() {
    console.log("Hello world!");
  }
}

class Employee extends Person {
  pin: number = 1234;
}

const employee = new Employee();

//{"name":"Bill GATES","pin":1234}
console.log(JSON.stringify(employee));

employee.sayHello();//Hello world!
```

Une fois qu'une classe a hérité d'une autre, il est possible de redéfinir une méthode de la classe parente dans la classe enfant. Il n'y a aucun mot clé nécessaire pour le faire, il suffit juste de réutiliser le même nom de méthode :

```
class Person {
  sayHello(): void {
    console.log("Hello from Person!")
  }
}

class Employee extends Person {
  sayHello(): void {
    console.log("Hello from Employee!")
  }
}

const person = new Person();
person.sayHello();//Hello from Person!
const employee = new Employee();
employee.sayHello();//Hello from Employee!
```

Autre détail intéressant, il est possible d'utiliser le mot clé « *super* » pour faire référence à la classe parente (le mot clé est identique à celui de Java). Il est d'ailleurs obligatoire de faire appel au constructeur de la classe parente si celui-ci définit des paramètres et que vous souhaitez avoir un constructeur dans la classe enfant. Le mot clé « *super* » trouve ici sa première utilité :

```
class Person {
  name: string;

  constructor(name: string) {
    this.name = name;
  }
}

class Employee extends Person {
  pin: number;

  constructor(name: string, pin: number) {
    //Obligatoire si on ne veut pas
    //avoir une erreur de compilation
    super(name);
    this.pin = pin;
  }
}

const employee = new Employee("Bill GATES", 1234);

//{"name":"Bill GATES","pin":1234}
console.log(JSON.stringify(employee));
```

Un dernier petit exemple pour finir cette première partie du dossier. Le mot clé super permet aussi, lorsque l'ont redéfinit une méthode, de faire référence à la méthode de la classe parente, vraiment très pratique :

```
class Person {
  getHello(): string {
    return `Hello i'm a person`;
  }
}

class Employee extends Person {
  getHello(): string {
    return `${ super.getHello() } and an employee!`;
  }
}

const employee = new Employee();
//Hello i'm a person and an employee!
console.log(employee.getHello());
```

Conclusion

Dans cette première partie nous nous sommes intéressés à l'historique de TypeScript, aux outils pour développer, et aux concepts de base permettant de démarrer avec le langage. Comme vous avez pu le voir il y a déjà beaucoup à dire et TypeScript est encore bien plus riche que ça. Dans la prochaine partie, je vais donc aller plus loin sur le langage et aborder des caractéristiques qui lui sont propres. Si TypeScript est JavaScript, l'inverse n'est pas vrai et le langage permet d'aller plus loin que le standard ECMAScript sur certains aspects.

Je profite aussi de cette première partie pour vous inviter, si par chance vous êtes à Paris et si vous êtes intéressé par TypeScript, à rejoindre la communauté sur Meetup (http://www.typescript.paris/). Nous avons la chance d'avoir un groupe communautaire actif avec des gens sympathiques, accessibles qui partagent leur passion pour le langage ! •

Michaël Bertocchi
Ingénieur développement
@dupot_org

Développer pour le mobile avec une technologie multiplateforme

PARTIE 3 : MA PREMIÈRE APPLICATION ANDROID AVEC QT

Dans le précédent article vous avez pu avoir une courte présentation de l'environnement de développement, nous allons ici développer une application Android en utilisant du Javascript et du QML, le C++ servira uniquement à instancier l'application. C'est parfait pour les personnes qui peuvent avoir des difficultés ou être "effrayées" par le C++, le Java et le C#. Notre application sera une simple todo List, vous verrez ainsi comment gérer des données persistantes, des changements d'écran et des évènements d'interactions. Le code de cette application est disponible sur Github :
https://github.com/imikado/programmezArticleTodoQtAndroid

Création de l'application

Ouvrez votre IDE Qt Creator pour créer une application « tutoProgrammez », choisissez le type Application > Quick Application. Lorsque vous commencez une application, ne vous précipitez pas. Essayez d'abord de prendre un papier, un crayon, ou une page de votre éditeur de texte préféré et écrivez succinctement les différents écrans. Voici ici un schéma de l'application qui sera développée :
Ecran "Menu":
• Bouton: Liste des tâches
Ecran "Liste des tâches":
• Boucle sur la liste des taches,
• Bouton pour ajouter une nouvelle tâche.
Popup formulaire :
• Chargement des données d'une tâche,
• Ajout d'une nouvelle tâche.

Créons le mode de changement d'écran

Dans une application mobile, la première partie à développer n'est pas l'interface principale, mais la mécanique de changement d'écran, ainsi, une fois ce socle installé vous pouvez plus facilement créer le menu et les différents écrans.
Pour cela, Qt met à votre disposition le composant *StackView*
Voici le code concerné, ajouté dans le fichier « main.qml »:

```
StackView {
    id: stack
    anchors.fill: parent
}
```

Grâce à l'ajout de ce composant, nous créons une fonction qui permettra de charger un fichier QML à l'écran :

```
function launch(sView){
    stack.push(sView);
}
```

Gestion des dimensions

Comme vous le savez, il existe désormais une multitude d'écrans mobiles, mais vous n'allez pas vous amuser à développer pour chaque. Pour cela je vous propose d'utiliser une fonction pour convertir les dimensions afin de les adapter à chaque écran.
Pour cela, nous ajoutons les propriétés de dimensions réelles, virtuelles ainsi qu'une variable indiquant si nous sommes en développement ou non. Ainsi, en « dev », c'est-à-dire en compilant sur notre ordinateur, on compilera avec la résolution virtuelle. Sinon on se retrouve avec un affichage paysage (lié à nos écrans 16/10 ème).

```
Window {
  property bool dev:true
  property int _width: 480
  property int _height: 800
  width:{ if(dev===true){ _width }else{ Screen.width } }
  height:{ if(dev===true){ _height }else{ Screen.height } }
```

Ainsi que deux méthodes toujours dans le fichier main.qml

```
  function calculate(value_){
    return value_*iRatio;
  }

  function init(){
    iRatio=(width/_width);
    launch('qrc:/page_menu.qml')
  }
```

Puis, au chargement de l'application nous indiquons les dimensions réelles de l'écran :

```
Component.onCompleted:init();
```

Enfin, à chaque appel à une dimension, nous utilisons cette fonction *calculate()*.

Création de la page de gestion des tâches

Nous ne regarderons pas le code du menu qui n'est pas très complexe, vous pouvez toujours y jeter un œil sur le dépôt Github.
Avec la gestion des tâches, nous entrons dans le vif du sujet, vous allez voir que Qt propose un moyen très simple de gérer des listes

d'éléments. Nous allons utiliser les concepts du « model binding »: nous créons un objet model, nous attachons des objets d'affichage QML, et à chaque évolution, l'affichage est synchronisé, vous verrez c'est magique. Tout d'abord dans le fichier « main.qml » nous créons un objet ListModel avec le code suivant :

```
ListModel{
    id:modelTasks
}
```

Voyons maintenant le code de la page de gestion des tâches « page_taches.qml » :

```
import QtQuick 2.0
import QtQuick.Layouts 1.3
import QtQuick.Controls 1.4
Rectangle {
  width:parent.width
  height:parent.height
  color:"#789598"
  ToolBar {
    height:calculate(140)
    RowLayout {
      width:parent.width
      height:parent.height
      ToolButton {
        text: qsTr(" < ")
        onClicked: stack.pop()
      }
      Label {
        text: "TO DOs"
        elide: Label.ElideRight
        horizontalAlignment: Qt.AlignHCenter
        verticalAlignment: Qt.AlignVCenter
        font.pixelSize: calculate(20)
      }
      Item { Layout.fillWidth: true }
      ToolButton {
        text: qsTr(" + ")
        onClicked: addTask()
      }
    }
  }
  ColumnLayout{
    y:calculate(180)
    width:parent.width
    spacing:calculate(20)
    Repeater{
      model:modelTasks
      Row{
        Layout.alignment: Qt.AlignCenter
        Rectangle {
          radius: calculate(10)
          width:calculate(430)
          height:calculate(40)
          color:{
            if(model.done===1){  "#7cc16f" }
            else { "#fff" }
          }
          Text{
            x:calculate(10)
            y:calculate(10)
            text:model.titre
            font.pixelSize:calculate(12)
          }
          MouseArea{
            anchors.fill: parent;
            onClicked:function(){ editTask( model.index ); }
          }
        }
      }
    }
  }
}
```

Regardons en détail les parties intéressantes :

- Le bouton "+" qui appelle la fonction addTask() pour ajouter une tâche,
- Le connecteur "branché" sur la couche ListModel qui permettra de boucler sur les tâches,
- L'affichage du fond de couleur en fonction du statut "fait" ou non.

Pour boucler sur les tâches on utilise un composant Repeater, on lui indique sur quel ListModel se connecter

```
Repeater{
        model:modelTasks
```

Tout ce qui vient après sera utilisé pour boucler sur chaque élément. Ensuite pour reprendre les données variables de l'élément bouclé on utilise le mot clé model, comme ceci (model.titre, model.index) :

```
  Text{
        x:calculate(10)
        y:calculate(10)
        text:model.titre
        font.pixelSize:calculate(12)
    }
    MouseArea{
        anchors.fill: parent;
        onClicked:function(){ editTask( model.index ); }
    }
```

Mais vous devez vous demander à quel moment on a défini que les tâches avaient un champ titre ou done. En réalité, on ne le fait pas, ou plutôt, on le fait à la volée (nous le verrons juste après).
J'ai fait le choix d'utiliser une popup pour ajouter et modifier une tâche. Les deux utilisent le même élément. Une propriété indiquera si on est en ajout ou en mise à jour.

```
function addTask(){
    popupTask.myIndex=-1;
    popupTask.action="insert";
    popupTask.open();
    myPopup.reset();
  }
```

Ce code sert à afficher la popup, son code dans le fichier « main.qml » correspond à ceci :

```
Popup {
    id: popupTask
    x: calculate(90)
    y: calculate(100)
    width: calculate(300)
    height: calculate(300)
    modal: true
    focus: true
    closePolicy: Popup.CloseOnEscape | Popup.CloseOnPressOutside
    property int myIndex:0
    property string action:"update"
    Mypopup{
      id:myPopup
    }
  }
```

Il est intéressant ici de voir la propriété action et l'appel implicite à un autre fichier QML « Mypopup.qml » :
Voir code complet sur **www.programmez.com**

Jetons un œil à la fonction saveTask() définie dans le fichier « main.qml »

```
function saveTask(){
    if(popupTask.action === "update"){
      modelTasks.get(popupTask.myIndex ).titre=myPopup.getInputText();
      modelTasks.get(popupTask.myIndex ).texte=myPopup.getTextarea();
      if(myPopup.getSwitchChecked() ){
        modelTasks.get(popupTask.myIndex ).done=1;
      }else{
        modelTasks.get(popupTask.myIndex ).done=0;
      }
    }else{
    modelTasks.append({titre:myPopup.getInputText(),texte:myPopup.getTextarea(),done:0});
    }
    popupTask.close();
  }
```

L'objet modelTasks est un ListModel qui contient une liste d'objets. Il est utilisé ici pour stocker des tâches.

```
modelTasks.append( {titre:myPopup.getInputText(),texte:myPopup.getTextarea(),done:0} );
```

Vous noterez que nous ajoutons simplement des objets au format Json, nul besoin de définir une structure au préalable.
Ainsi il est aussi simple par la suite d'y accéder via les propriétés.
A l'ajout d'une tâche via la popup, il apparaît instantanément dans la liste des tâches. Idem lorsque vous modifiez les propriétés d'une d'entre elles. Notez également qu'en modifiant le « flag » fait ou non, sa couleur change également. Nul besoin de rafraîchir, alors que la couleur est une simple propriété QML. Pour rappel : c'est magique.

```
color:{
    if(model.done===1){  "#7cc16f" }
    else { "#fff" }
  }
```

Ajoutons de la persistance à nos tâches

Dans l'état actuel, l'application fonctionne : vous pouvez ajouter une tâche, l'éditer, la flaguer comme faite, voire la supprimer ; mais dès que vous l'éteignez vous perdez tout, nous allons voir comment faire persister ces informations.
Pour cela, nous allons simplement ajouter une écriture dans une base sqlite, embarquée dans l'application.
Dans le fichier main.qml, il nous faut ajouter l'inclusion d'une nouvelle librairie :

```
import QtQuick.LocalStorage 2.0
```

Ensuite, nous allons modifier d'abord la fonction init pour créer la base et la table si elle n'existe pas :

```
function init(){
    _db = LocalStorage.openDatabaseSync("myTasksDb", "1.0", "My tasks", 1000000);
    _db.transaction(
          function(tx) {
            // Creation de la table si elle n'existe pas
            tx.executeSql('CREATE TABLE IF NOT EXISTS MyTasks(id INTEGER
 PRIMARY KEY,titre TEXT, texte TEXT, done INT)');
            var rs = tx.executeSql('SELECT * FROM MyTasks');
           //boucle sur les tâches stockuees en SQL pour les ajouter au model
 de l'application
            for (var i = 0; i < rs.rows.length; i++) {
              modelTasks.append({
                    titre:rs.rows.item(i).titre,
                    texte:rs.rows.item(i).texte,
                    done:rs.rows.item(i).done,
                    id:rs.rows.item(i).id
                  });
            }
          }
    );
    iRatio=(width/_width);
    launch('qrc:/page_menu.qml')
  }
```

Si'il y a des enregistrements, nous en profitons également pour boucler dessus afin de les ajouter à notre *ListModel*.
Notez au passage que l'on a ajouté un champ *id* qui permettra par la suite de pouvoir modifier et supprimer en base de données.
Nous faisons de même pour la modification et l'ajout de tâches dans la fonction *saveTask()*:

```
function saveTask(){
    if(popupTask.action === "update"){
      var oUpdateRow=modelTasks.get(popupTask.myIndex );
      oUpdateRow.titre=myPopup.getInputText();
      oUpdateRow.texte=myPopup.getTextarea();
      if(myPopup.getSwitchChecked() ){
        oUpdateRow.done=1;
      }else{
        oUpdateRow.done=0;
      }
      _db.transaction(
            function(tx) {
              // Create the database if it doesn't already exist
```

```
            tx.executeSql('UPDATE MyTasks SET titre=?,texte=?,done=? WHERE id=?',[
                    oUpdateRow.titre,
                    oUpdateRow.texte,
                    oUpdateRow.done,
                    oUpdateRow.id
                ]);
            }
            );
    }else{
     var oNewRow={titre:myPopup.getInputText(),texte:myPopup.getTextarea(),done:0};
      oNewRow.id=modelTasks.count;
      _db.transaction(
            function(tx) {
              // Insert tasks
              tx.executeSql('INSERT INTO MyTasks (titre,texte,done,id) VALUES (?,?,?,?)',[
                    oNewRow.titre,
                    oNewRow.texte,
                    oNewRow.done,
                    oNewRow.id
                ]);
            }
            );
      modelTasks.append(oNewRow);
    }
    popupTask.close();
}
```

Compilons

Lorsque vous developpez votre application, vous avez besoin de voir le rendu au fur et à mesure pour vérifier l'ergonomie générale ainsi que l'algorythme. Ici vous avez la chance d'avoir une technologie multiplateforme, je vous conseille de compiler en code natif (vers l'OS de votre ordinateur). Ainsi vous aurez une compilation très rapide pour avancer. Enfin, une fois que vous serez proche du but, vous pourrez compiler vers la platefome mobile.

Note

Malheureusement, vous ne pouvez compiler vers iOS (iPhone/iPad) qu'à partir d'un Mac.

Avant de compiler, vous devez choisir votre cible en cliquant sur l'icône au dessus du trinangle vert [figure 1]
Selectionnons par exemple "Desktop" pour compiler vers notre OS, ici Linux Mint.
Une fois sélectionné, vous avez le choix. Soit "juste" de compiler avec l'icône marteau : ceci vous génèrera un fichier exécutable de votre application. Soit de compiler puis d'exécuter l'application via le triangle vert.

Note

lors de la compilation, il génère dans un répertoire du même nom de votre projet avec pour suffixe la plateforme cible [figure 2].

Par exemple pour une compilation GNU/Linux, voici le contenu du repertoire [figure 3]

Sous Windows, le contenu sera similaire, sauf que vous aurez des .dll et un fichier .exe .
Pour Android, en revanche, lors de la compilation vous aurez une multitude de fichiers, mais les principaux se trouvent être :

- le fichier d'installation .apk,
- et le fichier xml de paramétrage (Android Manifest.xml) .

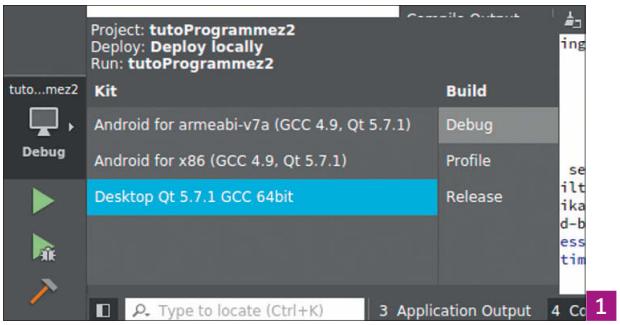
Note

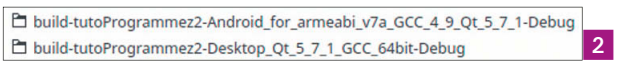
Le fichier .apk est l'équivalent d'un .msi sous windows, il permet en le lancant d'installer votre application et d'ajouter son icone sur le smartphone/tablette de votre choix. Vous pouvez installer votre apk sur celui-ci sans passer par une application de Store, à condition au préalable de l'avoir autorisé dans vos paramètres "autoriser l'installation d'application issue de sources inconnues" Par exemple, ce que je fais: j'active un serveur web sur mon ordinateur, j'y dépose le fichier .apk, et, ainsi, de n'importe quel tablette/smartphone, je me rends à l'url de mon ordinateur pour telecharger et installer l'application pour la tester.

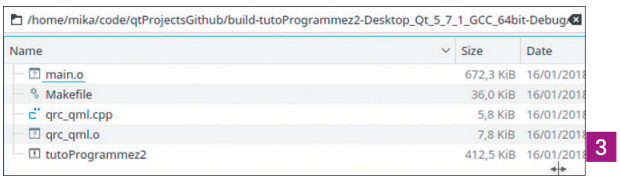
Conclusion

Comme vous avez pu le voir dans cet article il est assez simple d'écrire une application multiplateforme mobile avec cette librairie. De plus il est assez simple, même pour un débutant, de développer ces applications avec l'usage de Javascript et QML qui restent des langages très simples à prendre en main.
Enfin, rappelez-vous "Write once, run anywhere" : après avoir pris en main ce framework, vous pourriez également profiter de son côté multiplateforme pour vos programmes PC (Windows, MacOS et GNU/Linux).
J'espère que cet article vous donnera un autre éclairage sur les solutions disponibles concernant le développement multiplateforme ainsi que sur ce bel outil qui continue d'évoluer depuis des années.

•







Denis Duplan
Sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Utiliser l'interruption VERTB sur Amiga

Un dernier pour la route ! Nous continuons dans la foulée des précédentes séries d'articles. Ils prenaient pour prétexte la programmation d'une cracktro (en deux parties) et celle d'un sine scroll (en cinq parties), pour présenter dans le détail la manière d'attaquer le hardware de l'Amiga 500 en assembleur 68000 après avoir court-circuité l'OS. Cette fois, il convient de revenir sur un point délicat qui a été soulevé, à savoir celui de la gestion des « interruptions hardware ».

Le problème plus particulièrement étudié est le suivant : comment mettre sous « interruption VERTB » un bout de code dont l'unique fonction est de modifier la couleur de fond (COLOR00) ? Pour bien en visualiser les effets, nous allons élaborer un scénario au fil duquel la couleur de fond sera notamment modifiée par un tel bout de code quand le faisceau d'électrons atteint certaines positions verticales (Figure 1).

Vous pouvez télécharger le source du programme présenté ici à l'URL suivante : http://www.stashofcode.fr/code/interruption-vertb-sur-amiga/vertb0.s

Le fonctionnement des interruptions hardware

Comme cela a été expliqué dans un précédent article, le hardware de l'Amiga génère des événements qui peuvent entraîner des interruptions de niveau 1 à 6 du CPU.

Par exemple, l'événement VERTB, qui survient quand le faisceau d'électrons ayant atteint le bas de l'écran entame son retour vers le haut de ce dernier pour afficher la trame suivante, peut entraîner une interruption de niveau 3 du CPU. Le CPU cesse alors d'exécuter le programme principal pour exécuter le gestionnaire d'interruption dont l'adresse est stockée dans une table des vecteurs d'interruptions. S'agissant d'une interruption de niveau 3, le M68000 8-/16-/32-Bit Microprocessors User's Manual nous apprend que c'est le vecteur 27, qui renvoie au gestionnaire d'interruption logé à l'adresse $6C (Figure 2).

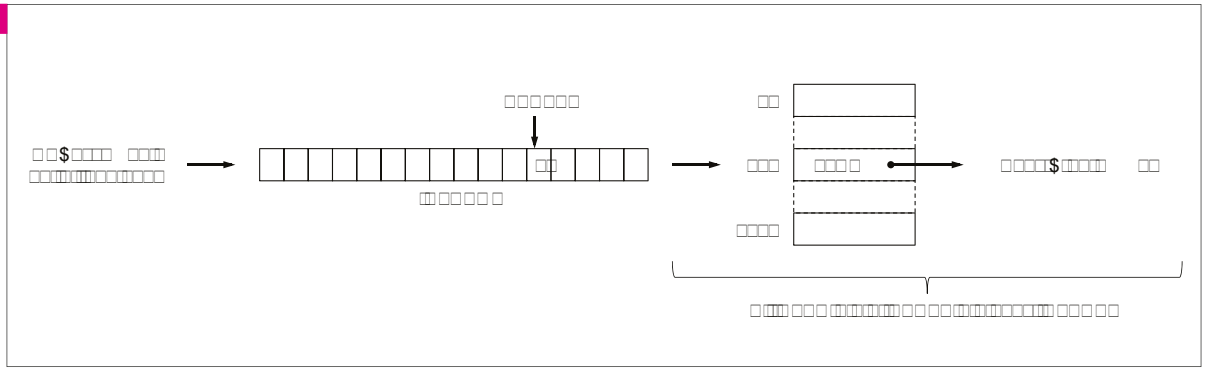
C'est donc par abus de langage qu'on parle d'interruption hardware, comme par exemple d'interruption VERTB pour désigner l'interruption dont il vient d'être question. Toutefois, cette convention étant pratique, nous l'utiliserons par la suite. La liste des évènements signalés le hardware qui sont susceptibles d'entraîner une interruption de niveau 1 à 6 du CPU figure dans la documentation du registre INTENA dans l'*Amiga Hardware Reference Manual* :

Bit	Nom	Niveau	Description
15	SET/CLR		Ce bit est décrit plus loin.
14	INTEN		Ce bit est décrit plus loin.
13	EXTER	6	Interruption externe
12	DSKSYN	5	Le contenu du registre DSKSYNC correspond aux données du disque
11	RBF	5	Buffer de réception du port série plein
10	AUD3	4	Bloc du canal audio 3 terminé
09	AUD2	4	Bloc du canal audio 2 terminé
08	AUD1	4	Bloc du canal audio 1 terminé
07	AUD0	4	Bloc du canal audio 0 terminé
06	BLIT	3	Opération au Blitter terminée
05	VERTB	3	Début du blanc vertical
04	COPER	3	Copper
03	PORTS	2	ports d'entrée/sortie et timers
02	SOFT	1	Réservé pour une interruption initiée par logiciel
01	DSKBLK	1	Bloc disque terminé
00	TBE	1	Buffer de transmision du port série vide

1

Changement de COLOR00 par le Copper, le programme principal et un gestionnaire d'interruption VERTB.

2



Appel du gestionnaire de l'interruption Level 3 Interrupt Autovector en cas d'évènement VERTB.

Le hardware signale **toujours** les évènements énumérés en positionnant les bits correspondants dans un autre registre, INTREQ. En revanche, il n'interrompt pas toujours le CPU dans la foulée. En effet, tout dépend de la gestion que le registre INTENA permet de configurer.

Lorsqu'une valeur 16 bits X est écrite dans INTENA, l'état du bit SET/CLR dans X indique si un bit d'INTENA désigné par un bit positionné dans X doit être positionné ou effacé. Par exemple, pour inhiber le déclenchement d'une interruption niveau 3 du CPU sur un événement VERTB, nous devons écrire `$0020` dans INTENA. A l'inverse, pour l'activer, nous devons écrire `$8020` dans ce registre. Toutefois, il faut encore compter avec le bit INTEN.

Le bit INTEN est une sorte d'interrupteur général :

- Lorsque INTEN est effacé, le déclenchement de toute interruption CPU sur évènement hardware est inhibé, quand bien même le bit correspondant à un événement est positionné dans INTENA. Par exemple, si jamais INTEN n'était pas positionné, écrire `$8020` dans INTENA ne suffit pas à activer le déclenchement d'une interruption de niveau 3 du CPU sur évènement VERTB. Il faudrait plutôt écrire `$C020` dans le registre pour positionner simultanément INTEN et VERTB, ou alors y écrire `$8020` pour positionner VERTB puis `$C000` pour positionner INTEN – l'inverse étant possible.
- Lorsque INTEN est positionné, le déclenchement d'interruptions CPU sur événements hardware est activé, mais uniquement pour les évènements dont les bits correspondants sont positionnés dans INTENA. Par exemple, écrire `$C000` n'a d'autre effet que d'activer le déclenchement d'une interruption de niveau 3 du CPU sur évènement VERTB, si jamais VERTB est le seul bit correspondant à un évènement à avoir été positionné dans INTENA.

Une conséquence importante de ce fonctionnement, c'est que nous pouvons écrire dans INTENA pour modifier la manière dont le hardware génère des interruptions CPU sur évènements hardware – la gestion des interruptions hardware pour faire plus court – de manière très sélective. Dans l'exemple suivant, la première instruction ne fait qu'activer l'interruption hardware TBE, et la seconde ne fait qu'inhiber l'interruption VERTB – tout ce qui concerne la gestion des autres interruptions hardware et l'interrupteur général INTEN est inchangé :

```
move.w #$8001,INTENA(a5)
move.w #$0020,INTENA(a5)
```

Pour cette raison, seule une lecture de INTENAR, le registre homologue de INTENA en lecture seule – INTENA est en écriture seule –, permet d'être exactement renseigné sur la gestion des interruptions hardware à un instant donné.

Couper les interruptions hardware

Nous n'allons pas rentrer dans le détail des opérations requises pour mettre en place un environnement de développement en utilisant WinUAE pour Windows et pour écrire un programme minimal en assembleur 68000 qui prend le contrôle total du hardware de l'Amiga. Tout cela a déjà été soigneusement exposé dans un article précédent.

D'emblée, concentrons-nous donc sur ce qui concerne la gestion des interruptions hardware. Après avoir court-circuité l'OS, il s'agit de prendre le contrôle des six vecteurs d'interruption du CPU qui, parmi les 255 dont ce dernier dispose, correspondent aux interruptions de niveau 1 à 6. Autrement dit, les vecteurs 25 à 30.

Tout d'abord, nous devons sauvegarder l'état de la gestion des interruptions hardware pour la restaurer à la fin. Pour cela nous lisons et stockons dans des variables `intena` et `intreq` le contenu des registres INTENAR et INTREQR, versions en lecture seule des registres INTENA et INTREQ :

```
move.w INTENAR(a5),intena
move.w INTREQR(a5),intreq
```

Nous pouvons alors couper toutes les interruptions hardware :

```
move.w #$7FFF,INTENA(a5)
```

Dans la foulée, nous acquittons tous les évènements que le hardware pouvait avoir signalés. Certes, le hardware va continuer à en signaler, mais du moins saurons-nous à partir de cet instant que si le hardware positionne un bit dans INTREQ, ce sera pour signaler un évènement survenu après que nous ayons pris la main :

```
move.w #$7FFF,INTREQ(a5)
```

Enfin, nous détournons les vecteurs des interruptions de niveau 1 à 6 en les renvoyant sur un sous-programme de type gestionnaire d'interruption qui nous appartient. Ces vecteurs pointent sur les adresses `$64` à `$78` :

```
lea $64,a0
lea vectors,a1
REPT 6
move.l (a0),(a1)+
move.l #_rte,(a0)+
ENDR
```

Plus loin, après le RTS final du programme principal, nous codons le gestionnaire d'interruption `_rte` en question :

```
_rte:
  rte
```

Ce gestionnaire ne fait donc strictement rien, pas même acquitter la notification de l'événement hardware qui est à l'origine de son appel – nous reviendrons sur ce point.

Un scénario à base d'interruptions VERTB

Comme expliqué, INTREQ fonctionne exactement comme INTENA, sauf qu'il sert à signaler des évènements plutôt qu'à activer/inhiber le déclenchement d'interruptions CPU associées aux évènements en question.

Il faut rajouter deux choses sur INTREQ :

- rien ne peut empêcher le hardware de continuer à signaler dans INTREQ les évènements qui surviennent ;
- il est possible d'écrire au CPU dans INTREQ pour simuler un ou plusieurs évènements hardware.

Nous allons illustrer ces points. Le scénario retenu est le suivant :

- le programme principal attend le quart supérieur de l'écran (ligne `DISPLAY_Y+(DISPLAY_DY>>2)`) pour déclencher une interruption VERTB dont le gestionnaire passe COLOR00 à `$00F0` (vert) ;

- la Copper list attend la moitié de l'écran (ligne `DISPLAY_Y+(DISPLAY_DY>>1)`) pour passer COLOR00 à `$0F00` (rouge) ;
- le hardware attend le bas de l'écran (fin de la ligne 312 en PAL) pour déclencher une interruption VERTB dont le gestionnaire passe COLOR00 à `$000F` (bleu).

La mise en code du scénario

Commençons la Copper list.
Cette dernière se résume à une instruction WAIT pour attendre que le faisceau d'électrons atteigne ou dépasse la mi-hauteur de l'écran, suivie d'une instruction MOVE pour stocker `$0F00` (rouge) dans COLOR00 :

```
move.l copperlist,a0
move.w #((DISPLAY_Y+(DISPLAY_DY>>1))<<8)!$0001,(a0)+
move.w #$FF00,(a0)+
move.w #COLOR00,(a0)+
move.w #$0F00,(a0)+
```

Bien évidemment, nous n'oublions pas de terminer la Copper list par le traditionnel `WAIT` impossible :

```
move.l #$FFFFFFFE,(a0)+
```

Codons maintenant le programme principal.
Tant que l'utilisateur n'a pas pressé le bouton gauche de la souris, ce programme attend en boucle le faisceau d'électrons au premier quart de hauteur de l'écran. Il stocke alors `$00F0` (vert) dans une variable `color` dans laquelle le gestionnaire d'interruption VERTB trouvera la couleur à stocker dans COLOR00. Enfin, il provoque l'appel à ce gestionnaire en générant une interruption VERTB par une écriture dans INTREQ :

```
_loop:

;Attendre le quart supérieur de la hauteur de l'écran

_wait0:
  move.l VPOSR(a5),d0
  lsr.l #8,d0
  and.w #$01FF,d0
  cmp.w #DISPLAY_Y+(DISPLAY_DY>>2),d0
  blt _wait0

;Déclencher une interruption VERTB

  move.w #$00F0,color
  move.w #$8020,INTREQ(a5)

;Tester la pression du bouton gauche de la souris

  btst #6,$bfe001
  bne _loop
```

Attention ! Ce code doit être complété par une boucle `_wait1` venant immédiatement après la boucle `_wait0` . Son objet est d'attendre le faisceau d'électrons à la ligne 0 :

```
_wait1:
  move.l VPOSR(a5),d0
  lsr.l #8,d0
  and.w #$01FF,d0
  bne _wait1
```

En effet, l'exécution du code de notre programme prend moins de temps qu'il n'en faut au faisceau d'électrons pour terminer de balayer la ligne `DISPLAY_Y+(DISPLAY_DY>>2)` . A défaut d'attendre une ligne suivante – en l'occurrence, la ligne 0, mais cela aurait parfaitement pu être la ligne du dessous `DISPLAY_Y+(DISPLAY_DY>>2)+1` –, le code du programme ne serait donc pas exécuté une fois par trame, mais plusieurs fois, ne produisant alors pas l'effet désiré.
Enfin, il nous reste à coder le gestionnaire d'interruption.
Avant toute chose, il faut se rappeler qu'un gestionnaire d'interruption est un sous-programme exécuté alors qu'un programme était en cours d'exécution. Partant, il faut au moins sauvegarder le contenu des registres qui seront utilisés par le gestionnaire pour être en mesure de les restaurer quand ce dernier se terminera. Cela s'effectue généralement à l'aide de l'instruction `MOVEM` pour stocker et lire le contenu de registres dans la pile.
Ensuite, il faut savoir que tant que l'événement n'est pas acquitté en effaçant son bit dans INTREQ, le hardware génère l'interruption CPU associée. Pour s'en convaincre, il suffit d'oublier d'acquitter l'événement VERTB dans le gestionnaire d'interruption et de modifier le programme principal pour lui demander de passer en boucle COLOR00 à `$0FF0` (jaune) :

```
_loop:
  move.w #$0FF0,COLOR00(a5)
  btst #6,$bfe001
  bne _loop
```

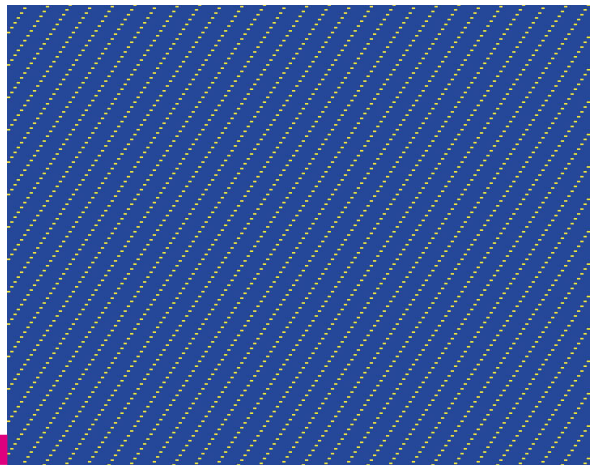
L'écran est totalement bleu ou presque, n'étant ponctué de jaune que lors des rares cycles d'exécution que la fin du gestionnaire d'interruption permet au CPU de récupérer pour continuer à exécuter le programme, avant qu'il ne doive exécuter de nouveau le gestionnaire d'interruption (Figure 3).
Dans ces conditions, le gestionnaire d'interruption doit impérativement acquitter l'évènement à la suite duquel il a été appelé.
Dès lors, le code de notre gestionnaire d'interruption doit au moins être le suivant :

```
_vertb:
;  movem.l d0-d7/a0-a6,-(sp)    ;Indispensable, mais à limiter aux registres modifiés
   move.w #$0020,INTREQR(a5)
;  movem.l (sp)+,d0-d7/a0-a6    ;Indispensable, mais à limiter aux registres modifiés
   rte
```

En l'espèce, son code complet est le suivant :

```
_VERTB:
;  movem.l d0-d7/a0-a6,-(sp)    ;Indispensable, mais à limiter aux registres modifiés
   move.w color,COLOR00(a5)
   move.w #$000F,color
   move.w #$0020,INTREQ(a5)
;  movem.l (sp)+,d0-d7/a0-a6    ;Indispensable, mais à limiter aux registres modifiés
   rte
```

La Copper list, le programme principal et le gestionnaire d'interrup-



3 *Le programme est presque court-circuité par le gestionnaire d'interruption qui n'acquitte pas VERTB.*

4 *Version plus complexe où le Copper déclenche une interruption VERTB.*

tion étant codés, il ne reste plus qu'à activer l'ensemble. Après avoir activé le Copper et lui avoir demandé d'exécuter en boucle la Copper list, il suffit de stocker l'adresse de notre gestionnaire d'interruption à l'adresse contenue dans le vecteur de l'interruption de niveau 3 du CPU, c'est-à-dire `$6C` … :

```
move.l #_VERTB,$6C
```

…avant d'autoriser de nouveau le hardware à générer cette interruption de niveau 3 quand il détecte l'évènement VERTB :

```
move.w #$C020,INTENA(a5)   ;INTEN=1, VERTB=1
```

Le résultat produit est celui présenté au début de cet article.

Restaurer les interruptions hardware

Lorsque l'utilisateur clique sur le bouton gauche de la souris, le programme se termine en rendant aussi proprement que possible la main à l'OS. Sans revenir sur les autres tâches qui nous incombent – elles sont détaillées dans un précédent article –, concentrons-nous donc encore sur ce que concerne la gestion des interruptions hardware.

Nous commençons donc pas couper de nouveau la génération d'interruptions CPU par le hardware… :

```
move.w #$7FFF,INTENA(a5)
move.w #$7FFF,INTREQ(a5)
```

…avant de restaurer les vecteurs d'interruption… :

```
lea $64,a0
lea vectors,a1
REPT 6
move.l (a1)+,(a0)+
ENDR
```

…et de rétablir la génération des interruptions CPU par le hardware :

```
move.w intreq,d0
bset #15,d0
move.w d0,INTREQ(a5)
move.w intena,d0
bset #15,d0
move.w d0,INTENA(a5)
```

C'est fini !

Une subtilité pour conclure…

Noter que le Copper a accès en écriture à INTREQ. Par conséquent, il est parfaitement possible de déclencher une interruption VERTB à n'importe quelle position du faisceau d'électrons que le Copper peut repérer – comme cela a été expliqué dans un article précédent, ses capacités en la matière sont limitées, la granularité verticale étant certes d'une ligne, mais la granularité horizontale étant de 4 pixels en basse résolution.

Il suffit donc d'utiliser un `MOVE` dans la Copper list :

```
move.l copperlist,a0
;...
move.w #INTREQ,(a0)+
move.w #$8020,(a0)+
```

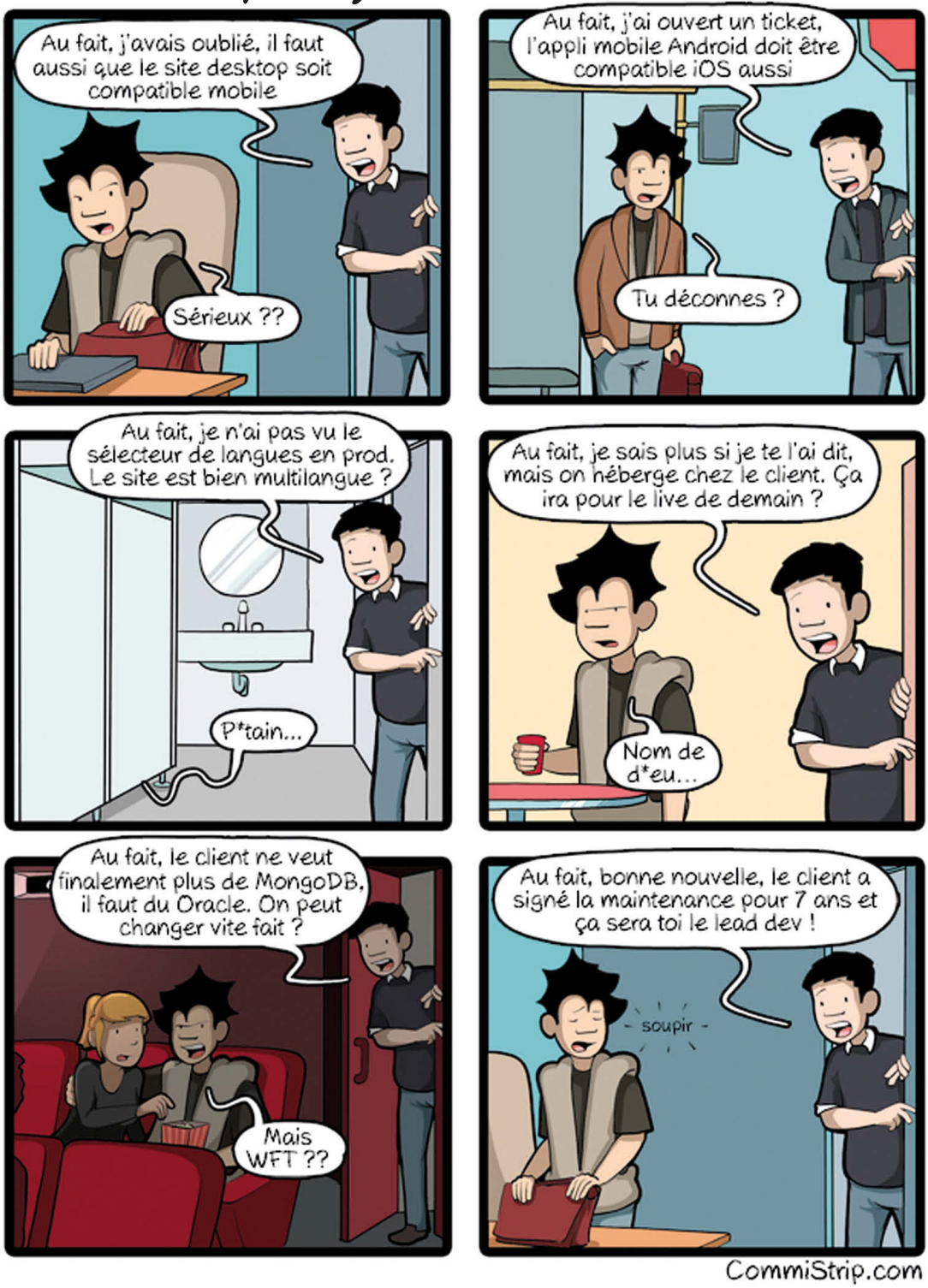
A titre, d'exemple, voici le résultat produit par une variante du programme présenté plus tôt (Figure 4). Ici, le Copper déclenche une interruption VERTB quand le faisceau d'électrons atteint ou dépasse le dernier quart de la hauteur de l'écran, avec pour effet de passer COLOR00 à `$0000` (noir). •

Vous pouvez télécharger le source du programme qui produit cet effet à l'URL suivante :
http://www.stashofcode.fr/code/interruption-vertb-sur-amiga/vertb1.s

Liens utiles

Amiga Hardware Reference Manual :
http://amigadev.elowar.com/read/ADCD_2.1/Hardware_Manual_guide/node0000.html
M68000 8-/16-/32-Bit Microprocessors User's Manual :
http://www.nxp.com/assets/documents/data/en/reference-manuals/M68000PRM.pdf

Quand tu penses que tu en as fini avec ce projet alors qu'en fait, PAS DU TOUT !



Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication & Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie
Ont collaboré à ce numéro : S. Saurel
Nos experts techniques : M. Reynaud, L. Ellerbach, S. Dolcini, P. Ficheux, R. Rigot, L. Devolder, C. Pichaud, J. Demageon, A. Chopard, S. Pontuseau, M. Bertocchi, D. Duplan, C. Prudent, C. Villeneuve, S. Bovo, E. Duport,
Couverture : © Lenovo - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - 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
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, avril 2018

Ce numéro comporte un encart jeté PC Soft pour les abonnés.

Abonnement :
Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy, 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com*
Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30.

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
35 € (monde entier) souscription sur www.programmez.com

tangente l'aventure mathématique

Choisissez votre abonnement :
PLUS avec les HS « kiosque »
SUPERPLUS avec la Bibliothèque Tangente

Mieux comprendre le monde grâce aux mathématiques

Politique, économie, finance, jeux, musique, littérature, arts plastiques, architecture, informatique, physique, biologie, géographie… Les mathématiques sont partout !!! Le magazine *Tangente* et ses hors séries vous aident à redécouvrir notre quotidien.

La version numérique pour tous

Il est maintenant possible de consulter sans limitation de durée les numéros et hors séries de *Tangente* en ligne sur le site **tangente-mag.com** pour tous ceux qui prennent l'abonnement numérique ou une des formules d'abonnement papier. Un véritable site interactif et pas un simple PDF. L'abonnement permet aussi d'accéder gratuitement aux problèmes du « Monde » sur **www.affairedelogique.com**

• • • • • Tangente Éducation

Trimestriel qui, pour un coût symbolique, traite de thèmes pédagogiques variés, donne des outils aux enseignants **et permet l'accès à de nombreuses ressources Internet.**
Nouvelle version numérique interactive !

Les hors séries « kiosque » • • • •

4 fois par an, un hors série (format « kiosque »)
Ces numéros d'au moins 56 pages explorent un grand dossier de savoir ou de culture.
Derniers parus : Les angles - Les graphes - Les démonstrations - Les fonctions - Maths des assurances - Maths et médecine - La droite - Maths et architecture - Complexes - Les ensembles - Maths et économie - Découpages et pavages - Espaces vectoriels
Disponibles :
– sur **www.infinimath.com/librairie**
– chez votre marchand de journaux
– **avec l'abonnement PLUS ou SOUTIEN.**

Les hors séries « Bibliothèque » • • • •

Pour nos lecteurs les plus curieux, les articles des hors séries de *Tangente* sont repris et complétés dans les livres de la Bibliothèque Tangente, magnifiques ouvrages d'environ 160 pages (prix unitaire 22,00 € à partir du 58), richement illustrés, disponibles
– sur la librairie du site **www.infinimath.com**
– chez votre libraire
– ou **en souscription avantageuse avec l'abonnement SUPERPLUS ou SOUTIEN.**

NOUVELLE FORMULE DE PAIEMENT !
Choisissez **l'abonnement permanent** par prélèvement.
Vous êtes débité **tous les 6 mois** de la somme indiquée dans la colonne "Permanent" en arrêtant quand vous voulez après deux ans.

Bulletin d'abonnement valable jusqu'au 31–08–18

À retourner à : Espace Tangente – service abonnement – 2 rue de la Prée – 27170 COMBON

NOM* . . . . . PRÉNOM* . . . . .
ADRESSE* . . . . .
CODE POSTAL* . . . . . VILLE* . . . . . PAYS* . . . . .
MAIL* . . . . . PROFESSION . . . . .

codif : Programmez

Abonnement Nos formules	Papier + Numérique			Numérique seul			Supplément papier hors métropole		
	1 an	2 ans	Permanent	1 an	2 ans	Permanent	1 an	2 ans	Permanent
ABO TG PLUS (6 nos + 4 HS par an)	☐ 65 €	☐ 124 €	☐ 30 €	☐ 42 €	☐ 80 €	☐ 19 €	☐ 25 €	☐ 50 €	☐ 12,5 €
ABO TG SUPERPLUS (par an, 6 nos + 4 livres Bibliothèque + 4 HS numérique)	☐ 102 €	☐ 198 €	☐ 48 €				☐ 30 €	☐ 60 €	☐ 15 €
ABO SOUTIEN : 6 nos de *Tangente* + 4 HS « kiosque » + 4 « Bibliothèque » + 4 nos de *Tangente Éducation*.	☐ 134 €	☐ 258 €	☐ 64 €				☐ 40 €	☐ 80 €	☐ 20 €

MONTANT TOTAL : ☐ €

MODE DE PAIEMENT

❒ Chèque (ordre Éditions POLE, uniquement payable en France) ❒ Bon de commande administratif

❒ Prélèvement : je recevrai mon mandat SEPA par mail et le renverrai signé

IBAN : ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

BIC : ☐☐☐☐ ☐☐☐☐ ☐☐☐

❒ Carte bancaire numéro ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ Date d'expiration ☐☐/☐☐

Cryptogramme ☐☐☐ **DATE :**

SIGNATURE :

MERCI

WINDEV TECH TOUR 2018 - 12 VILLES - 12 SUCCÈS