programmez.com

[Programmez!]

Le magazine des développeurs

216 MARS 2018

Visual Studio Code

L'outil gratuit de Microsoft

© Vertigo3d

Choisir une base de données

SQL, NoSQL, les critères, les usages : nos conseils !

Angular 5.0

Quelles nouveautés ?

Live Coding

Le développeur devient DJ !

Réalité virtuelle

C'est facile avec BabylonJS et Angular !

M 04319 - 216 - 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

EDITO

La donnée : objet informatique non identifié

Je me souviens d'une autre vie dans laquelle,quand je parlais données avec des développeurs, l'atmosphère se refroidissait. La donnée, et pire, la base de données, n'était pas forcément un objet d'une grande popularité. Il fallait l'utiliser, l'implémenter, se connecter à des bases. On parle aujourd'hui de l'opposition Devs et Ops mais n'oublions pas l'aversion entre développeurs et DBA. Certains DBA avaient l'accusation facile : c'est la faute du développeur si les accès aux données sont lents. C'était avant l'explosion du web et du mobile !

L'explosion des données a rebattu les cartes et, aujourd'hui, elles sont au coeur des apps, des sites web. La donnée est l'or noir des entreprises. Et le développeur a su la dompter même si cela n'a pas été sans douleur. Mais ce n'est pas forcément lui qui va manipuler les données les plus importantes. Le Big Data a introduit de nouveaux profils moins techniques mais plus analystes et statisticiens : le data scientist. Et on parle de données non structurés, de stockages objets, de bases à la demande, de données froides et chaudes, de Data Lake, d'architectures lambda et post-lambda. Les IoT génèrent des To de données qu'il faut ingérer, découper, stocker puis analyser. Et ce n'est qu'une infime volumétrie des données générées chaque jour.

"Serons-nous réduits à des (nos) données ?"

La donnée s'est transformée en objet technique du quotidien, et le développeur la voit comme un composant comme un autre qu'il faut développer, implémenter, gérer. Mais ce n'est pas pour autant qu'il devient un DBA.

Plus que jamais, la donnée est au coeur du monde d'aujourd'hui et de demain. L'intelligence artificielle a besoin de données pour apprendre et réagir à son environnement. L'IoT n'aurait que peu d'intérêt sans les données générées. Mais ces données brutes ont à la fois aucune et beaucoup de valeur. Elles peuvent indiquer le comportement des utilisateurs, être monétisées (avec ou sans notre accord), adapter un IoT, un service d'IA selon les données. Pour pouvoir être cet or noir, la donnée doit être traitée, analysée, interprétée. Bref, avoir la bonne donnée au bon moment et en extraire la bonne information.

François Tonic
ftonic@programmez.com

Qui êtes-vous ?
Je suis le nouveau numéro 2
Qui est le numéro 1
Vous êtes le numéro 6
Je ne suis pas un numéro. Je suis un homme libre.
(Le Prisonnier, 1967)

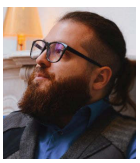
SOMMAIRE

Tableau de bord ---- 4
Agenda ---- 6
Abonnez-vous ! ---- 7
Live coding ---- 8
Retour sur CA World 2017 ---- 12
Matériel ---- 16
L'impasse de OS ---- 17
Angular 5 ---- 18


Développer pour le mobile ---- 20

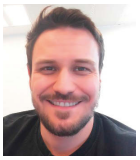

Visual Studio Code ---- 23

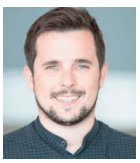



Choisir sa base de données ---- 29

Blockchain en Java ---- 47
Microsoft Teams ---- 53
IA Partie 2 ---- 55
OfficeJS ---- 57
VR + BabylonJS ---- 60


Ecrire une bibliothèque en Java ---- 64



FPGA ---- 67


La programmation orientée objet en C++ ---- 73


Vintage sur Amiga 500
Partie 4 ---- 78

CommitStrip ---- 82

Dans le prochain numéro !
Programmez! #217, dès le 30 mars 2018

CHOISIR UN MOTEUR 3D
Unity vs Unreal Engine vs CryEngine vs three.js vs Babylon.js vs Ogre vs Irrlicht.

DOSSIER TESTS LOGICIELS
Le développeur les néglige encore trop souvent.
Quels sont les différents types de tests ? Comment les utiliser ?
Les tests sont-ils solubles dans le DevOps et l'agilité ?

La police chinoise

va être équipée de lunettes à reconnaissance faciale.

Les Progressive Web Apps

seront-elles les apps de demain et vont-elles tuer les applications natives ?

Un journal national japonais sera présenté par un **androïde**.

Les **sites non HTTPS** seront marqués "Non sécurisé" par Google à partir de juillet prochain.

« Les talents français des métiers technologiques, qui sont pourtant reconnus comme ayant une formation universitaire et un savoir-faire de qualité, sont sous-payés par rapport à leurs homologues travaillant dans les autres capitales de la tech dans le monde. »

explique Antoine Garnier-Castellane, Directeur du bureau français de Hired.

Une partie du code source du **bootloader** d'iOS 9 s'est retrouvé sur GitHub début février. Ce morceau de code avait déjà fait un tour du reddit sans être vu mais là, l'affaire est tout autre. La question est de savoir comment un tel code a été mis en ligne et par qui ? Et comment il est sorti des serveurs d'Apple.

Tesla tente d'augmenter la cadence de production du Model 3 qui accumule les problèmes et les retards. 1 550 voitures ont pu être livrées sur le 4e trimestre 2017, sur les 4 100 attendues. L'objectif est d'atteindre une cadence de 5 000 voitures par semaine soit environ 260 000 voitures par an. Cette estimation est bien moins ambitieuse que les 500 000 voitures annuelles promises en 2016…

Quels sont les langages les plus utilisés par les utilisateurs de GitHub ?

Le classement va faire pleurer certains développeurs ! Ni Java, ni C ou C++ mais JavaScript et Python ! PHP arrive 6e, C# 7e et Go accroche la 9e place ! Les 3 premiers représentent 50 % des langages utilisés.

Rank	Language	Monthly Active Users	Trend
1	JavaScript	22.63%	
2	Python	14.75%	
3	Java	14.01%	
4	C++	8.45%	
5	C	6.03%	
6	PHP	5.85%	
7	C#	5.03%	
8	Shell	4.85%	
9	Go	4.10%	
10	TypeScript	3.89%	

Android KTX : du mieux pour Kotlin

Le nouveau langage d'Android, Kotlin, prend peu à peu ses marques. Android Studio, l'IDE de Google, intègre le langage dans la version 3. Pour aider les développeurs, Google propose Android KTX, des extensions pour écrire plus rapidement du code Kotlin.

Kotlin :

```
sharedPreferences.edit()
.putBoolean("key", value)
.apply()
```

Kotlin with Android KTX :

```
sharedPreferences.edit {
    putBoolean("key", value)
}
```

Pour en savoir plus : https://github.com/android/android-ktx/

Un nouveau framework Microsoft : Blazor

Blazor est un framework expérimental d'interfaces utilisateur web. Il est basé sur les technologies C#, Razor, et HTML. Il s'exécute dans les navigateurs modernes supportant WebAssembly, ce qui promet des performances décoiffantes à l'exécution, proches du code natif. Razor et navigateur (Browser) donnent son nom à ce framework : Blazor. Il propose toutes les fonctionnalités attendues d'un framework moderne, dont :

- un modèle composant pour la création des interfaces utilisateur,
- le routage,
- des gestionnaires de mises en formes (layouts),
- interopérabilité avec JavaScript,
- formulaires et validations,
- débogage aussi bien dans le navigateur que dans un environnement de développement (EDI),
- etc.

Site : https://github.com/aspnet/blazor

QUI CONTRIBUE RÉELLEMENT À L'OPEN SOURCE ?

Ce titre d'un article d'InfoWorld a attisé notre curiosité. Qui sont les contributeurs aux projets open source ? Une fois de plus, GitHub est la source d'informations, même si la plateforme n'est pas l'unique lieu de contribution. Les données brutes fournissent des indications intéressantes mais n'expliquent pas la stratégie des grands contributeurs. Attention : tous les éditeurs n'ont pas 50 000 employés et des milliers de développeurs. Donc il faut garder en tête les échelles de grandeurs entre les très gros éditeurs et les petits.

Le top 10 est le suivant :

Editeur	Nombre de développeurs
1 Microsoft	4 550
2 Google	2 267
3 Red Hat	2 027
4 IBM	1 813
5 Intel	1 314
6 Amazon.com	881
7 SAP	747
8 ThoughtWorks	739
9 Alibaba	694
10 GitHub	676

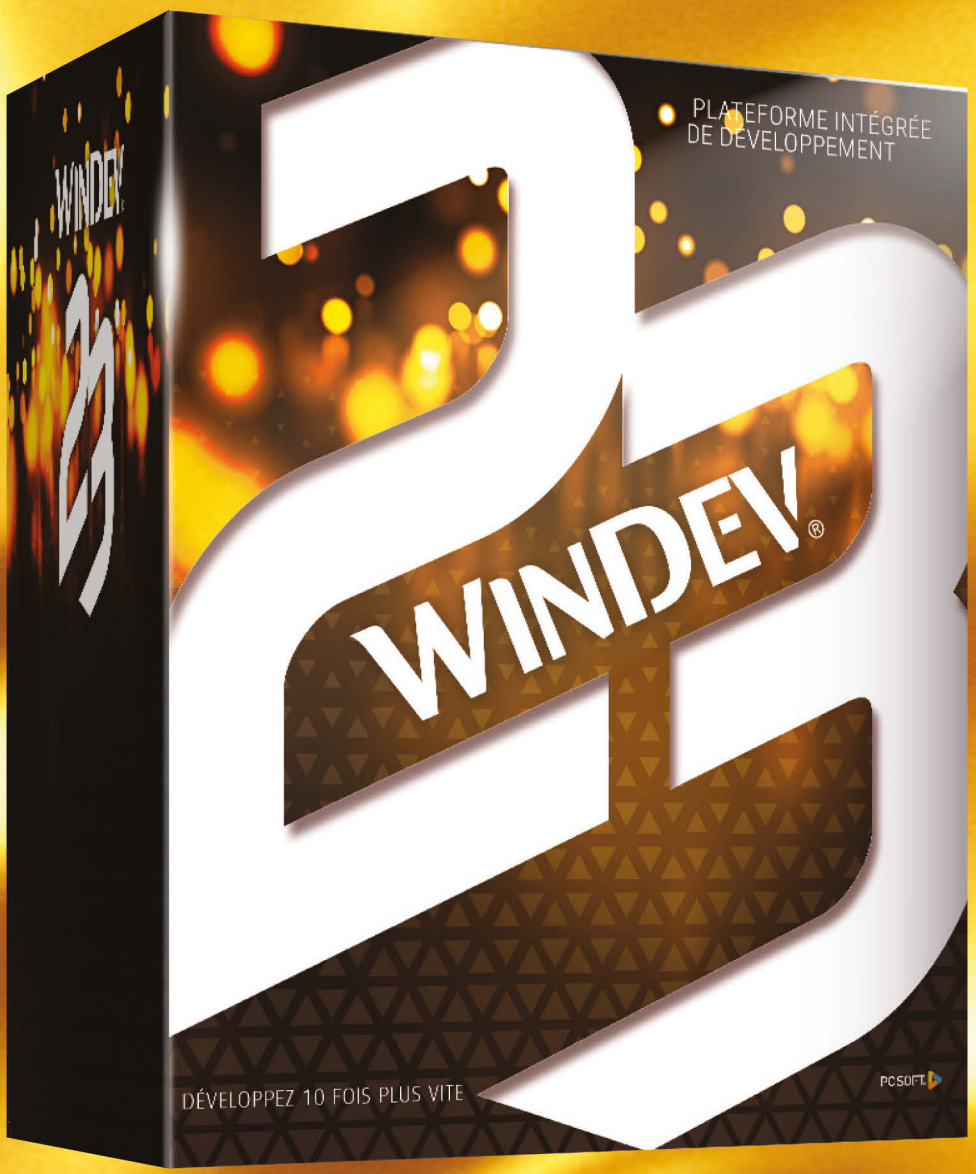
On constate que Microsoft est le plus gros fournisseur de contributeurs, Google et Red Hat suivent à bonne distance. On notera aussi l'absence de plusieurs éditeurs importants dans ce top 10 : Facebook (11), Pivotal (13), Mozilla (16), Oracle (17), Suse (23), Apple (25)…

Le **bug bounty** rapporte ! Google a distribué 2,9 millions $ pour traquer les vilains bugs dont 1 million pour les failles Android ! Depuis 2010, Google a versé 12 millions $.

WINDEV 23

12 VILLES
DU 13/3 AU 12/4

Tech Tour



COMMUNAUTÉS

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

GDG Toulouse
17 mai : WebVR et intégration continue.

GDG Code d'Armor
20 mars : Java 9 & 10 en 60 minutes chrono !

ParisJUG
13 mars : JEE
15 mai : spécial 10 ans de ParisJUG !

Lyon JUG
7 mars : spécial DDD

MARS

Microsoft Tech Summit : 14 & 15 mars – Paris
Participez à des workshops gratuits, animés par la crème des ingénieurs à l'origine de nos solutions Cloud sur Azure et Microsoft 365. Grâce au Microsoft Tech Summit, enrichissez vos compétences, approfondissez votre expertise et allez plus loin dans votre carrière professionnelle.

LLVM, Clang, lldb, lld, Polly : 27 mars - Paris
Le prochain meetup de la communauté se tiendra fin mars. Le thème n'est pas encore connu.

AVRIL

Add Fab : 11 & 12 avril – Paris
Pour sa deuxième édition, Add Fab rassemblera les 11 et 12 avril 2018, Porte de Versailles, les acteurs les plus importants de l'industrie de l'Impression 3D ou Fabrication Additive. Regroupant les sociétés les plus représentatives et novatrices du secteur, de la Start-up à l'entreprise internationale, afin de présenter une offre exhaustive du secteur : logiciels, imprimantes, prototypage, matériaux, équipements, outillage et formation. En parallèle se déroulera un cycle de conférences, d'ateliers et de formations en direct.

MakemeFest Angers
Le nouveau salon maker reviendra en avril à Angers. L'événement 2017 avait réuni +70 startups et makers, + 3000 visiteurs. Pourquoi pas vous ?
Site officiel : http://makeme.fr

Oracle Cloud 2018
Oracle aime les développeurs et veut le prouver avec cette journée 100 % code, 100 % développeur. L'événement se déroulera dans plusieurs villes dans le monde dont Paris. Nous y trouvons des sessions, des ateliers. La date française n'est pas encore connue.

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

EclipseCon 2018
Les 13 et 14 juin, la conférence EclipseCon se tiendra une nouvelle fois à Toulouse. L'appel à conférence se terminera le 19 mars. N'hésitez pas à proposer des sessions :
https://www.eclipsecon.org/france2018/

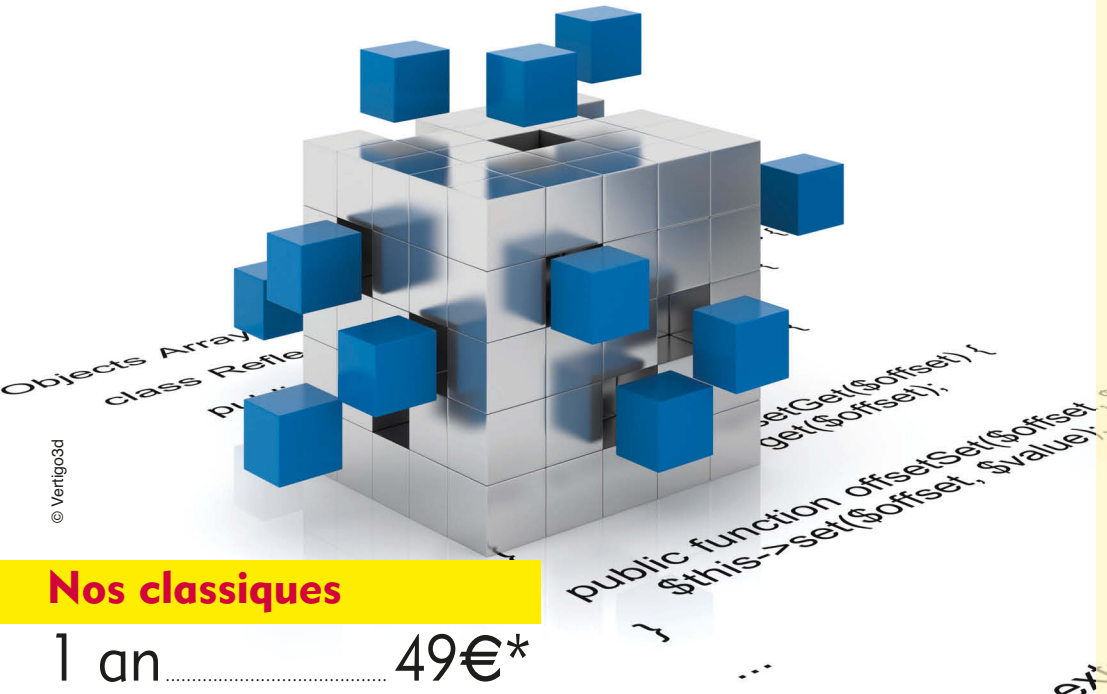
DevFest Lille
Le Devfest Lille 2018 se passera le 21 juin 2018 sur une seule journée pour 400 personnes prévues. La conférence prendra place dans les locaux de IMT Lille-Douai (20 Rue Guglielmo Marconi, 59650 Villeneuve-d'Ascq). Le site est accessible via https://devfest.gdglille.org/
Le CFP (http://devfestlille.cfp.io/) est en cours jusqu'au 1 avril 2018 (pour un programme disponible début mai)

Hack in Paris
Du 25 au 29 juin, Hack in Paris est la grande conférence sur le sécurité et le hacking en France.

NE RATEZ AUCUN NUMÉRO

Abonnez-vous !

[Programmez!] Le magazine des développeurs



Nos classiques

1 an ..... 49€*
11 numéros

2 ans ..... 79€*
22 numéros

Etudiant ..... 39€*
1 an - 11 numéros

** Tarifs France métropolitaine*

Abonnement numérique

PDF ..... 35€
1 an - 11 numéros
Souscription uniquement sur
www.programmez.com

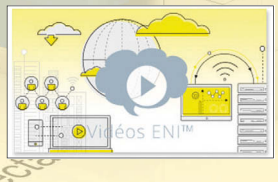
Option : accès aux archives 10€

Nos offres d'abonnements 2018

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

- Arduino*
  Apprenez à programmer votre microcontrôleur
- jQuery*
  Maîtrisez les concepts de base

eni Editions



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

- Arduino*
  Apprenez à programmer votre microcontrôleur
- jQuery*
  Maîtrisez les concepts de base

Offre limitée à la France métropolitaine
** Valeur de la vidéo : 34,99 €*

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 : Arduino
☐ Vidéo : jQuery

PROG 216
Offre limitée, valable jusqu'au 30 mars 2018

☐ Mme ☐ M. Entreprise : Fonction :

Prénom : Nom :

Adresse :

Code postal : Ville :

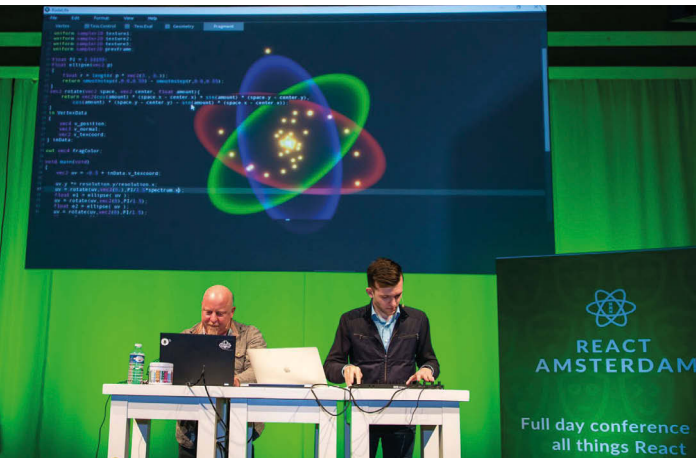
email indispensable pour l'envoi d'informations relatives à votre abonnement

E-mail : @

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

*** Tarifs France métropolitaine**

Bruce Lane
Développeur créatif (Cinder, openFrameworks, OpenGL)



Live-coding : la poésie de l'ingénieur

Coder en direct est une pratique populaire pour développeurs passionnés de musique et/ou de graphisme. Il s'agit de montrer le résultat en temps réel de la saisie de chaque lettre ou ligne devant une audience de tous horizons. Ceci pour démocratiser le processus tout en se faisant plaisir. Le code et le résultat sont vidéoprojetés et écoutables.

Ça rend public le processus de coder. Cela rend aussi la pratique moins élitiste tout en démocratisant les outils de création de logiciels dans le but de transmettre sa passion.

Tout ceci se déploie dans une ambiance « maker », sans se prendre au sérieux. C'est aussi un désir de pousser l'ordinateur à ses limites et découvrir des nouveaux usages, en détournant les pratiques habituelles. Une improvisation en temps réel, avec ses risques et ses moments magiques : partir sur une idée de son, échouer en essayant de la coder, et se retrouver avec une sonorité beaucoup plus intéressante à explorer ! Souvent l'artiste apprend davantage que le public. La démarche créative est différente du jeu sur un instrument avec ses limites physiques, dans lequel on adapte une idée à l'instrument ; en live-coding, les idées se développent pendant l'expérimentation. Sortir de son bureau en s'imposant un challenge ; enfin seul avec ses propres faiblesses et son génie, sans chef de projet ou utilisateurs dans les pattes et s'exposer aux risques du débogage en temps réel. Coder est un art qui prend toute sa dimension en live ; comment captiver le public progressivement tout au long de la performance ? La démarche est de facto open source à travers la collaboration ; la communauté s'entraide pour l'apprentissage de langages et techniques diverses afin d'arriver à un résultat rapidement audible/visible.

Le clavier

Dans mon cas, saisir sur un clavier d'ordinateur avec deux doigts est lent, le temps de taper une ligne (dans le noir de surcroît) je perds beaucoup de temps, ce qui est interminable en situation live, au risque de perdre l'attention du public. Pour l'anecdote, à ma première performance de livecode de shader à Live Performers Meeting, il m'est arrivé de taper un caractère de trop sans m'en apercevoir, scroller vers le bas, taper d'autres lignes qui ne s'exécutaient pas à cause de l'erreur introduite plus haut, et pour couronner le tout le texte était blanc sur fond de shader très lumineux, impossible de retrouver l'erreur ! De plus, le clavier AZERTY n'est d'ailleurs pas très adapté pour la vitesse de frappe avec ses touches composées (AltGr 4 pour une accolade…) et ses accents, par rapport au QWERTY. Un clavier rétroéclairé peut aider, d'ailleurs...

Et pourquoi pas utiliser le Stenophone (https://github.com/jarmitage/stenophone): 1

Langages et logiciels

Selon vos goûts, une pléthore de langages est à votre disposition.

Les langages procéduraux avec une syntaxe de style C comme OpenGL sont pointilleux avec les accolades, points virgules, etc. Bien que standards et bien connus, difficile de coder rapidement sans erreur en partant de zéro.

Une technique consiste à mettre en commentaire et décommenter au fur et à mesure. Une autre est de créer en amont des fonctions qu'il suffit d'appeler au moment venu.

Intéressant aussi d'introduire une erreur volontairement et de la corriger au bon moment pour se synchroniser avec le premier temps de la mesure musicale. Un exemple de live-coding visuel de WebGL dans le navigateur avec The_Force de Shawn Lawson : 2

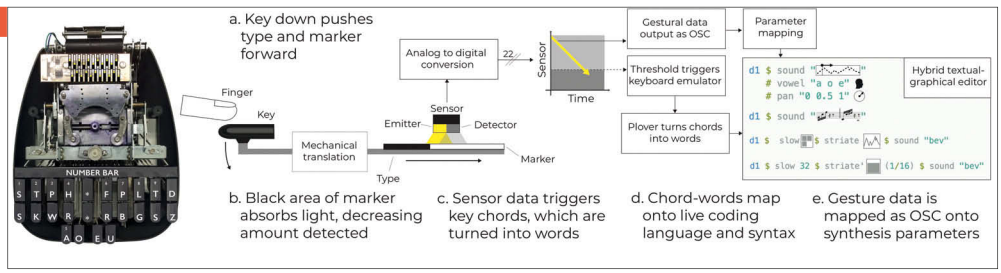
Chaque pixel du canvas exécute ce programme en parallèle sur le GPU! Si la syntaxe est correcte, le shader compile et s'affiche. La compilation a lieu à chaque caractère saisi, l'affichage reste le même tant que l'erreur n'est pas corrigée. La ligne 2 normalise (0.0 à 1.0) les coordonnées de l'écran pour le vecteur uv (x, y) en divisant les coordonnées d'entrée par la résolution du canvas. La ligne 3 effectue le rendu à l'écran d'un vecteur (rouge, vert, bleu, alpha), le résultat est étonnant, non ? À voir en live : http://www.shawnlawson.com/portfolio/liveware/

Dans le domaine de la programmation OpenGL, Kodelife, créée par Rob Fischer a.k.a. Hexler est une application desktop : 3

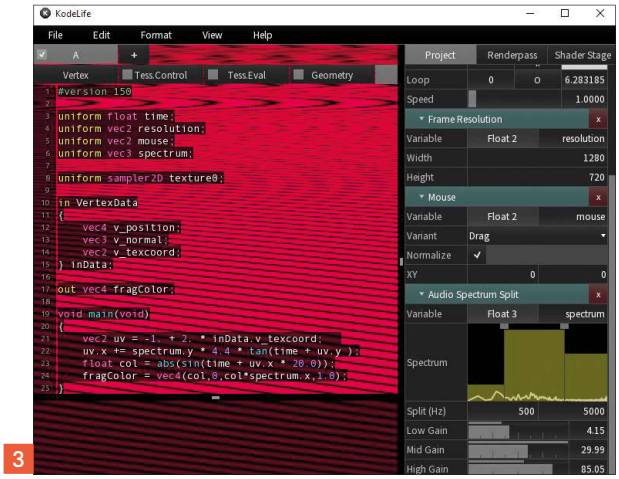
Kodelife affiche les valeurs des variables disponibles (appelées uniform) sur la partie droite de l'interface utilisateur. Nous avons la position de la souris mouse(x, y), l'analyse du spectre audio sous forme d'un vec3 spectrum(x : basse, y : medium, z : aigu), le temps depuis le lancement du programme time(millisecondes). Vous pouvez donc vous replonger dans vos livres de maths, car ici nous animons la scène avec des sinus, tangentes, valeurs absolues !

Mais saisir toute cette syntaxe avec ces parenthèses, accolades et points-virgules est fastidieux, essayons un autre site Internet dans le navigateur avec Hydra d'Olivia Jack : 4

1

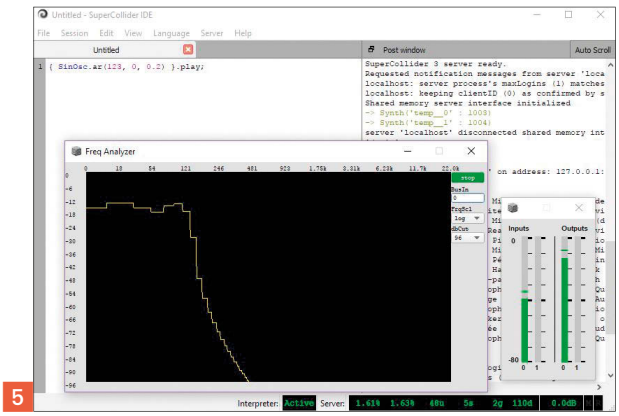


2

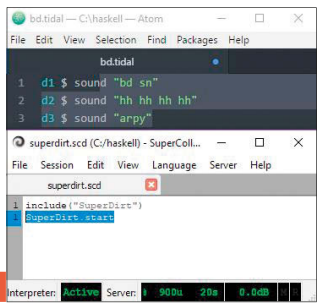


3

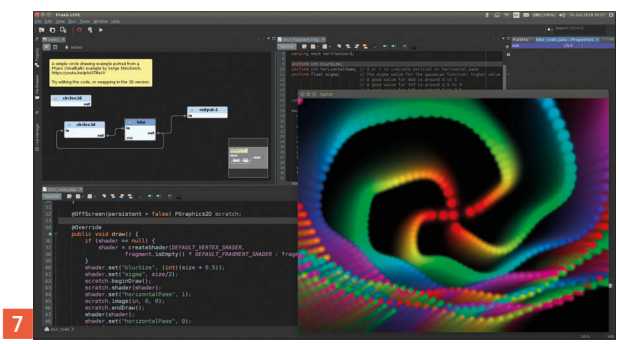
4



5



6


7

Du WebGL sous le capot, le code est basé sur une syntaxe JavaScript moderne plus efficace, au détriment d'un code standard, s'approchant de la programmation fonctionnelle. Ici le code exécuté avec Ctrl-Entrée (ligne) ou Shift-Entrée (le tout). « o0 » est un des 4 buffers de rendu, avec un dégradé chaîné à une rotation en fonction du temps. Les fonctions de la librairie masquent la complexité de WebGL.

Côté musique, pas le premier mais le précurseur, Super Collider en 1996, pose les bases du mouvement avec une interface puissante. Il utilise son propre langage, sclang (dérivé du C et de Ruby/Scala).

Il faut d'abord lancer le serveur dans le menu Server/Boot server, saisir son code, ici une sinusoïde, puis appuyer sur Ctrl-Entrée pour écouter le résultat. 5

D'autres environnements se sont construits sur la base de Super Collider, comme Overtone (écrit en Clojure), Lua2SC (lua), mais aussi Tidal Cycles (haskell). 6

Tidal Cycles tire parti de l'éditeur Atom avec un plugin. Nous avons ici créé un rythme et un synthé sur 3 pistes (d1 à d3). Nous obtenons un séquenceur, les pistes sont jouées parallèlement avec le synthé SuperDirt, qui étend les sons de base de Super Collider.

La programmation fonctionnelle en Haskell est pertinente et plus efficace que la syntaxe C. Praxis LIVE combine le code avec la programmation visuelle 7

À la manière d'un synthétiseur analogique, on « patche » des modules ensemble, puis on étend les fonctionnalités par du code Java ou OpenGL. J'ai vu une performance de Neil C. Smith, le créateur de Praxis LIVE au festival Generate, utilisant son logiciel en mode musique, avec un sample du fameux Amen Break. Il a joué 30 minutes de création musicale avec ce seul sample, tordu, allongé, découpé, c'était incroyable !

Un autre logiciel qui allie la programmation visuelle au code est Fugio : 8

Multi-plate-forme, ici sur Raspberry Pi 3, Fugio combine plusieurs langages, comme OpenGL et Lua, entres autres. Grâce aux possibilités de partage de texture entre applications avec Spout sous Windows ou Syphon sur macOS, le résultat visuel peut être envoyé à une autre application de VJing par exemple, comme HeavyM (excellent logiciel de mapping français).

Si vous avez un Raspberry Pi 3, vous avez dû voir Sonic Pi préinstallé sur le système d'exploitation Raspbian. Voici un exemple de code produisant aléatoirement des notes de la gamme pentatonique avec un effet de type reverb :

```
with_fx :reverb, mix: 0.2 do
  loop do
    play scale(:Eb2, :major_pentatonic, num_octaves: 3)
.choose, release: 0.1, amp: rand
    sleep 0.1
  end
end
```

Attention toutefois avec ces logiciels audio, on peut vite arriver à créer des sons horribles (aigu, saturation), au risque de détruire du matériel de diffusion ou des oreilles… Il est préférable de bien maîtriser son code avant une performance.

Ressources

Le site TOPLAP https://toplap.org est une référence sur le Live-Coding, vous y trouverez les événements à venir, les actualités de la communauté et la possibilité de s'inscrire aux discussions de groupe. Une compilation de ressources assez complète est maintenue sur: https://github.com/lvm/awesome-livecoding/blob/master/README.md

Évènements

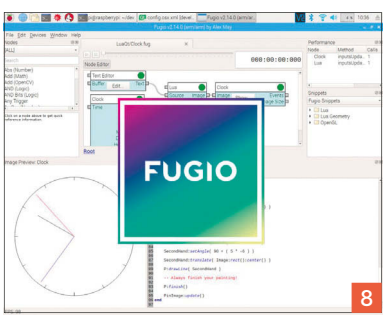
Des « Algorave » ont lieu régulièrement en streaming sur Internet, la prochaine : Algosix, du 15 au 17 mars : http://algorave.com/ actuellement en cours d'élaboration, suite à l'engouement pour le live-coding, les organisateurs pensent à rajouter un jour pour une cinquantaine de performances en plus!

Le réseau AVnode.net propose des festivals Audiovisuels (Live Performers Meeting/Rome – Generate – Tübingen, Splice – Londres, etc.) en Europe avec des performances et ateliers au sujet du live-coding.

Conclusion

Choisissez l'application et surtout le langage qui vous plaît le plus et essayez !

Un dernier lien pour l'inspiration, mon entrée lauréate pour le concours du W3C: https://videodromm.com/w3cContest/



8

ikoula
HÉBERGEUR CLOUD

PRÉSENTE

CLOUD**IKOULA**ONE

Le succès est votre prochaine destination

MIAMI SINGAPOUR PARIS
AMSTERDAM FRANCFORT

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

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

ikoula
HÉBERGEUR CLOUD

NOM DE DOMAINE | HÉBERGEMENT WEB | SERVEUR VPS | SERVEUR DÉDIÉ | CLOUD PUBLIC | MESSAGERIE | STOCKAGE | CERTIFICATS SSL

Yves Grandmontagne
Envoyé spécial à CA World 2017, Las Vegas

CA Technologies : CA World 2017, le mainframe et The Modern Software Factory

Evoquer CA Technologies, c'est évoquer le mainframe, un marché historique sur lequel l'éditeur continue d'investir. Mais ce serait ô combien réducteur, car CA Technologies est en train d'assurer sa mutation, s'imposant avec sa stratégie The Modern Software Factory en acteur incontournable de la gestion et du cycle de vie des applications, des API, des micro-services, et de la sécurité des développements.

Mike Gregoire, CEO de CA Technologies, lors de son keynote d'ouverture

CA Technologies est née il y a 40 ans. Premier éditeur à atteindre le milliard de dollars de chiffre d'affaires, expert reconnu dans le domaine du mainframe, il a longtemps défrayé la chronique avec sa stratégie agressive d'acquisitions, avant de sombrer dans l'incertitude, marquée en particulier par les errances sur son nom. Computer Associates devient CA et se noie dans l'internet, où un nom en deux lettres est trop faible pour être trouvé. Changement de direction, CA devient CA Technologies, et l'accent est mis sur l'économie des applications.

Bien lui en a pris, même si le virage, aujourd'hui encore, est difficile à prendre. Le mainframe est un monde à part, un monde vieillissant et de récurrence, qui apporte son chiffre d'affaires qui certes ne progresse plus, mais qui reste important et globalement se maintient d'une année sur l'autre. La migration des équipes d'une situation de rente à une activité concurrentielle est toujours difficile, et CA Technologies n'est pas à l'abri d'un revirement.

Mike Gregoire, CEO de CA Technologies>

Cependant, la migration semble bien engagée. C'est ainsi que le portefeuille applicatif de l'entreprise - qui a réalisé un chiffre d'affaires supérieur à 4 milliards de dollars et dans le même temps investi 1,9 milliard dans les développements organiques et les acquisitions – est composé de quatre lignes de produits : Agile, DevOps, Sécurité et Mainframe. Le mainframe n'est donc pas oublié, ni dans le discours ni dans les annonces, mais il ne fait plus recette auprès de la majorité des visiteurs de CA World 2017, venus chercher autre chose. Pour autant, CA Technologies a clairement un projet : apporter plus d'intelligence au mainframe ! Donc une seule motivation, automatiser, car dans 5 ans le mainframe aura perdu la majorité de ses compétences, partie à la retraite…

The Modern Software Factory

Justement, cet 'autre chose', c'est dans sa vision de l'écosystème du développement d'applications qu'elle s'exprime pour CA avec sa nouvelle stratégie *The Modern Software Factory*. Celle-ci repose sur quatre piliers : l'agilité de l'entreprise (*Business Agility*), l'automatisation et l'Intelligence Artificielle (*Intelligent Automation*), les compétences et les connaissances (*Experience Insights*), et la sécurité (*End-to-End Security*).

- L'**agilité** s'exprime dans la planification et la modernisation de l'architecture applicative au profit de la rapidité de production des applications (*time to market*) dans des entreprises de plus en plus numériques où les applications jouent un rôle de premier plan.
- L'**automatisation du développement**, de DevOps, des tests (dans un environnement de tests en continu) et des mises à jours pour plus de vélocité et de qualité au travers de l'élimination des processus de test et de mise à jour manuels, tout en accélérant l'adoption de l'innovation via des touches de plus en plus sensibles d'Intelligence Artificielle (IA) et de machine learning (ML).
- Les **insights**, la valorisation des données de l'entreprise du mobile au mainframe, destinés à l'amélioration constante via les analytiques au profit de la performance globale et des API dans un contexte d'expérience, et pouvant aboutir à la saisie de nouvelles opportunités.

Sur abonnement ou en kiosque

Le magazine des pros de l'IT

Mais aussi sur le web

L'INFORMATICIEN
www.linformaticien.com

- La **sécurité**, omniprésente de bout en bout, à la fois pour sécuriser les applications, les données et les infrastructures, pour réduire les risques et instaurer la confiance. Un leitmotiv : évoluer du DevOps vers le DevSecOps.

12 nouveaux produits ou nouvelles versions

Les annonces faites lors de CA World 2017, qui viennent enrichir un portefeuille applicatif déjà bien rempli de CA Technologies, participent et renforcent très clairement cette vision stratégique dont les effets relatés par les premières entreprises qui l'on adoptée sont significatifs. 1

Le tableau résume la vingtaine d'annonces qui ont rythmé CA World 2017. L'évolution des solutions existantes est moins visible, mais sensible. Par contre, une grande partie des nouveautés qui figurent au catalogue proviennent, c'est un phénomène récurrent chez CA Technologies, des acquisitions. Elles sont au nombre de 5 au cours de la période qui a précédé l'évènement :

- **Veracode** pour l'évaluation de la sécurité et la détection des risques dans le code ;
- **Blazemeter** pour l'optimisation de la performance du test des API dans la démarche DevOps ;
- **Runscope** pour le monitoring des API, qui vient compléter les solutions CA API Management et CA API Gateway déjà appréciées ;
- **Automic** pour l'automatisation des processus, applications et infrastructures, l'orchestration, ou encore la gestion des workloads et des releases ;
- Le mainframe n'est pas oublié avec **zIT Consulting**, une société allemande plus spécialement sur l'optimisation des coûts sur IBM System Z.

17 nouvelles solutions annoncées lors de CA World 2017

Agile Management	Security	Dev Products (APIM)	DevOps	Mainframe	Automation
ENHANCEMENTS	ENHANCEMENTS	NEW	NEW	NEW	ENHANCEMENTS
CA Agile Central CX Enhancements	CA Veracode Greenlight	CA Microgateway	CA BlazeMeter API Test	CA Trusted Access Manager for Z	CA Automic Workload Automation 12.1
CA PPM 15.3	CA Veracode Mobile Application Security Testing		CA Continuous Delivery Director SaaS	CA Dynamic Capacity Intelligence (zIT Consulting acquisition)	CA Automic Release Automation 12.1
			CA Digital Experience Insights		CA Automic Service Orchestration 12.1
			ENHANCEMENTS	ENHANCEMENTS	
			CA BlazeMeter	CA Mainframe Operational Intelligence 2.0 (New standalone offering)	
			CA Service Virtualization 10.2 (Includes SV Community Edition & CodeSV)		
			CA Test Data Manager 4.3 (includes CA Virtual TDM)		
			CA APM 10.7		

1

Au service de l'App Economy

Au cours du keynote d'ouverture, Mike Gregoire a rappelé l'importance des applications et de leur mise en œuvre rapide pour les entreprises.

Concrètement, cela se traduit chez CA par l'un des catalogues les plus complets dédié non au développement – les solutions et langages de programmations sont considérés comme des 'commodities' - mais à la gestion du cycle de vie des applications. Ce qui se traduit par des points forts comme Agile, DevOps, IA et machine learning toujours plus présents, une forte implication dans l'*API economy*, ou encore l'incontournable sécurité avec un engagement marqué sur la sécurité du code et le testing, et un fort accent mis sur l'automatisation qui tire profit de ce qui précède.

Autre axe important de la stratégie de CA, la volonté d'accompagner les organisations dans leurs choix numériques stratégiques, sachant que ceux-là se traduisent systématiquement par des infrastructures, même virtuelles, des applications (donc du code), et des API pour que tout le monde communique. Sans oublier les analytiques, analyse des données et des risques, également alimentés par l'IA. Voilà qui réserve à CA et sa *Modern Software Factory* une place de choix au coeur des stratégies numériques des entreprises et de leurs équipes de développement. •

OTTO BERKES, CTO DE CA TECHNOLOGIES : THINK FAST, SÉCURITÉ ET BLOCKCHAIN

Pour le directeur technique et 'tête pensante' de CA Technologies, nous assistons à une transition de l'entreprise numérique vers l'entreprise intelligente. « *L'âge des techniciens arrive. Nous devons transformer l'entreprise et sa culture pour favoriser la créativité individuelle* ». Cette vision se traduit par une expression qui revient régulièrement dans son discours : *'Think Fast'*. « *L'entreprise doit penser à son architecture dans 10 ans, composée de conteneurs et de micro-services, qui exploite le potentiel de l'IA pilotée non plus par la valeur mais par les stories. Le challenge du développeur, c'est d'aller plus vite* ».

Un bémol cependant dans cette vision, le 'security nightmare'. « *Sur la sécurité, rien n'a changé. Nous devons changer le paradigme et mettre la sécurité partout !* ». La blockchain entre dans cette stratégie, Otto Berkes entend l'utiliser sur le mainframe, non pas pour le Bitcoin (on s'en serait douté !), mais comme couche de sécurité pour la gouvernance de la donnée.

Otto Berkes, un conseil à la DSI ?

« *Elle doit être le leader des technologies et de leur orchestration, opérer plus en architecte pour produire et délivrer des applications. L'IT devient un système d'engagement et plus seulement un système d'enregistrement, qui génère des data pour générer des insights. Sans oublier que l'humain doit rester une part de l'équation. C'est pourquoi la séparation entre DSI et métiers doit se réduire. A la DSI de saisir l'opportunité du 'serverless computing' pour apporter de la valeur, s'assurer d'accéder à un large panel de données, se connecter transversalement aux multiples utilisateurs et industries, et d'injecter de l'IA et de l'automatisation* ».

Otto N-Berkes

Retour d'expérience

Eurosport : la gestion des API au coeur des nouvelles expériences des téléspectateurs

Eurosport a expérimenté l'enrichissement en temps réel des données sur le mobile et le web lors de la diffusion de courses cyclistes. Un exemple concret de l'exploitation des API.

Les téléspectateurs des courses cyclistes sur Eurosport, à commencer par le Giro, ont pu vivre une expérience nouvelle d'association de données sur le web et les applications mobiles pour enrichir l'évènement sportif retransmis en direct à la télévision.

En plus de suivre la course, ils ont pu suivre les coureurs, individuellement et en temps réel, disposant d'un accès à des données comme la localisation GPS, la position, la vitesse, l'altitude, et jusqu'à l'état du coureur, celui-ci étant équipé de capteurs. Une approche nouvelle dans le cyclisme – moins dans le sport en général, les voitures de F1 ou les rugbymen en sont déjà équipés - qui intéresse tant les fans de vélo que les amateurs, qui peuvent ainsi vivre au plus près l'expérience des coureurs, comprendre les stratégies de course et les échappées, mesurer les efforts humains...

Reposant sur un focus entre l'IoT et le Big Data, cette expérience exploite les outils de transmission des informations, de broadcast, de GPRS law frequency, déployés sur les courses. Les informations sont collectées sur un stockage Amazon, et exploitées par des applications. Toute la complexité du projet provient de la diversité des ressources... Ainsi que des négociations sur les droits des données ! Ainsi, chaque course est unique, et doit être négociée par la chaîne, auprès des organisateurs comme des équipes.

La vraie difficulté technique provient de la multiplication des sources, des réseaux qui leurs sont associés, des fichiers et de leurs formats, ou encore des applications pour les traiter. C'est là qu'interviennent les API. Eurosport s'est tournée vers CA Technologies, et sa plateforme de gestions des API et de mapping des sources et des apps - CA API Management et CA App Experience Analytics -, pour intégrer de manière transparente les API de chaque fournisseur de données via des services web en lien avec les bases de données et pour exposer les flux. La plateforme a permis de simplifier la gestion de la couche des API et de réduire sa complexité. Elle a également permis de traiter les formats de fichier en production et de faciliter la conversion des policies dans la gateway en approche low code.

Abreuvé de données accessibles sur un deuxième ou troisième écran (mobile, ordinateur…), le téléspectateur vit une expérience nouvelle. Avec pour Eurosport des résultats sensibles, une augmentation de 26% de l'audience, et un ROI rapide sur la publicité. « *La donnée n'a d'intérêts qui si on l'accompagne d'usages* », affirme-t-on chez Eurosport, qui entend renouveler et étendre l'expérience. Ce qui ne sera pas toujours simple, par exemple sur le Tour de France, la technologie devra répondre à d'autres challenges, en particulier la consommation des batteries et l'explosion des volumes de données à transmettre, stocker, et analyser... •

PAUL PEDRAZZI, GENERAL MANAGER DESIGN DE CA TECHNOLOGIES : COMPRENDRE ET OFFRIR UNE MEILLEURE EXPÉRIENCE

Paul Pedrazzi, General Manager Design

Dans le passé, l'objectif du design était de rendre les interfaces sympas. Acte souvent considéré comme mineur dans le processus de développement, le travail du designer intervenait après la réalisation des applications. Aujourd'hui, l'expérience utilisateur est devenue une priorité, avec des attentes qui se rapprochent des applications signées Apple, Google, ou Amazon. Les utilisateurs prennent des décisions d'achat et demandent de meilleurs logiciels. Le designer doit comprendre les produits et les utilisateurs, et travailler sur chaque interface afin d'offrir une meilleure expérience.

CA Technologies a fait le choix de se doter d'une division design unique, avec des équipes – « *The team is key* », nous a commenté Paul Pedrazzi - dédiées aux lignes de produits qui travaillent en proximité des développeurs, ainsi que des API et des interfaces. Le design intervient en amont des produits finis, auxquels il apporte de la valeur. C'est d'autant plus important que le mode SaaS se développe, imposant de tester les produits avec un design qui réponde aux objectifs des ventes, du marketing, etc., pour des entreprises emportées par la vague de la transformation numérique. Car il ne faut pas oublier que le logiciel est le support de cette transformation - et 30 % des applications auraient déjà basculé dans le cloud - et à ce titre il mérite toute l'attention des entreprises. « *Nous voulons rendre le numérique plus compréhensible pour les utilisateurs* », affirme Mike Gregoire.

Comment mesurer le ROI du design ?

« *Le ROI est indirect, dans le nombre d'utilisateurs et le volume des applications. Le budget de ma division va chez les hommes et dans les procédures d'estimation, dans le DevOps et dans les tests d'utilisabilité qui sont effectués sur tous les produits. Une interface n'est validée que si elle recueille 75% de validation.* »

Est-ce que les API et les microservices imposent de nouvelles contraintes ?

« *Il faut définir quelles sont les activités clés sur chaque produit. Et il faut automatiquement une interface pour le comprendre et manipuler. Sans oublier la performance qui est importante.* »

Un conseil ?

« *Pour construire un grand produit, il faut sortir du bureau, s'assoir à côté des utilisateurs et repérer ce qui fait sens. Et il faut créer quelque chose que les gens aiment. Je suis très motivé par la création des changements, le progrès est très motivant.* »

François Tonic

Texas Instrument dévoile un nouveau matériel pour l'éducation

TI continue à étoffer sa gamme de matériels éducatifs. Après un prometteur TI-Innovator Hub, c'est maintenant au TI-Innovator Rover d'être proposé. Dommage que ce Rover, tout comme le Hub, soit limité au monde de l'éducation, car il a toute sa place en dehors.

Faisons tout d'abord le tour du Rover. Le châssis est bien construit et solide. Il ne va pas se casser facilement. Il possède 3 ports d'entrée / sortie (via le Hub), d'un connecteur 20 broches pour le Hub, de deux connecteurs I2C, d'un mini-USB pour les données provenant de la calculatrice ou du logiciel TI-Nspire CX. Il est construit sur une puce MSP-EXP.

Par défaut, il possède un capteur de distance, d'un gyroscope et d'un capteur RGB. On peut déjà faire des manipulations basiques avec ces éléments. Mais comme on a besoin du Hub pour fonctionner, vous pouvez l'étendre avec de nombreux capteurs de type Seeed Grove. Et là, vous ouvrez de nombreuses possibilités. L'accéléromètre n'est pas proposé dans cette première version, mais le constructeur nous a précisé que ce capteur pourrait être intégré dans une prochaine version du Rover. Pour utiliser le Rover, il vous faut :

- le Rover (ça aide) ;
- l'Innovator Hub ;
- une calculatrice compatible (83 Premium CE, 84 Plus CE / CE-T, Nspire CX / CX CAS) ;
- les logiciels TI-Nspire et TI-Innovator Hub Software.

Les +
- **Ludique**
- **Programmation**
- **Les capteurs**
- **Fabrication**
- **Variété des scénarii**

Les –
- **Intégration matérielle pas optimale**
- **Couche logicielle pas totalement intégrée**
- **Trop de câbles différents**
- **Ergonomie éditeur**

Avant de démarrer

Avant toute chose, vous devrez mettre à jour la calculatrice et l'Innovator-Hub. Par exemple, sur la Nspire CX CAS, il faut disposer de la version 4.5 (minimum). Pour le Hub, installer la mise à jour la plus récente du Sketch. Cette étape est indispensable sinon, vous ne pourrez pas utiliser le Rover.

Le montage des éléments est simple :

- on connecte la nappe sur le connecteur du hub (arrière) et au connecteur du rover (dessous);
- on connecte le câble I2C sur le port I2C du Hub et au port I2C du rover (port de gauche);
- on branche le câble micro-USB sur le port de la calculatrice et le hub.

Le port USB de la Nspire CX est placé tout en haut et non plus sur le côté ce qui tend le câble. À terme, cela peut endommager le câble et les ports.

Pensez aussi à recharger les batteries de la calculatrice et du rover avant de les utiliser.

Voilà. Votre rover est opérationnel.

Une programmation basique

Deux solutions pour coder le Rover :

- directement sur la calculatrice via l'éditeur de programme ;
- depuis le logiciel desktop puis on transfère le code : pratique pour le clavier, mais cela oblige à déconnecter la calculatrice du rover.

La structure du problème est simple : le nom du programme puis début et fin du programme proprement dit :

```
Define rover()=
Prgm
  …
EndPrgm
```

Si on veut utiliser le rover, on commence par initialiser la connexion avec la commande Send "CONNECT RV" puis on enchaîne les commandes. On accède facilement aux commandes via bouton menu ->Hub -> 7 Rover (RV). De là, on peut avoir accès aux différentes fonctions du Rover : moteurs, capteurs. On peut détecter un tracé grâce au capteur de couleur placé sous le rover. Finalement, on s'habitue au codage sur le petit écran de la Nspire CX CAS, même si les gros doigts ne font pas forcément ami avec le clavier…

Crtl + R exécutera le code.

La documentation technique de base est le Ti-Innovator Technology Guidebook. On remarquera aussi des trous sur côtés du rover. Ils permettent d'y fixer des briques pour y installer des capteurs supplémentaires.

Notre avis

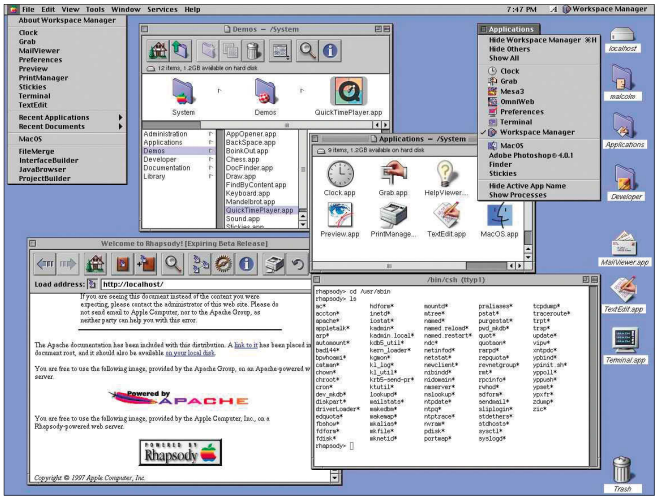
Globalement, ce Rover est bien construit et résistant. Sa prise en main est plutôt rapide et sa programmation pose peu de souci après quelques heures d'apprentissage et pour connaître les commandes disponibles. L'ergonomie de l'éditeur de code est sommaire et son usage sur la calculatrice énerve un peu. Dommage aussi que la plateforme ne vérifie pas la version logicielle du Hub.

Dommage que TI ne cible que l'éducation avec ce kit qui pourrait séduire bien au-delà, surtout, si le constructeur ajoute le support d'autres IDE, outils de programmation. Au moment d'écrire cet article, nous n'avions pas de prix tarif officiel. Une offre enseignante est cependant proposée à 119 . Au printemps prochain, des livres dédiés devraient être disponibles et de nouveaux tutoriels seront proposés. Aujourd'hui, les tutos manquent. •

François Tonic

L'impasse des OS

Depuis des mois, il y a une certaine frénésie autour des systèmes d'exploitation. Fin 2017, nous avions fait un point sur les différentes pistes actuelles chez Microsoft, Apple et Google : https://www.programmez.com/actualites/2018-andromeda-os-fuchsia-marzipan-26915



Rhapsody

Ces projets, même si tous n'ont pas été officialisés, n'agissent pas au même niveau. Andromeda était une première approche de modularité du système selon le terminal utilisé. Marzipan serait une couche pour unifier le modèle applicatif entre iOS et macOS. Fuschia est un système global pour unifier les différentes plateformes Google, utilisant DART, Vulkan et Material Design.

Quand on y repense, le système d'exploitation n'a pas forcément beaucoup évolué, dans ses concepts fondamentaux depuis l'apparition de l'Apple Lisa et du Macintosh. C'était il y a 35 ans ! La multiplication des plateformes matérielles (IoT, desktop, mobile) l'évolution constante des systèmes, la complexité croissante pèsent forcément sur l'OS. Aujourd'hui, un éditeur va avoir 2, 3 ou 4 systèmes, reposant plus ou moins sur les mêmes fondations, mais avec des modèles applicatifs peu ou pas compatibles. Et plus vous avez de systèmes, plus le travail de maintenance et de développement devient complexe.

Rapprocher les couches basses ne signifie pas forcément une unification totale autour d'un seul et unique système car parfois il y a un écart abyssal entre un desktop, un mobile, une voiture ou encore un IoT. Les capacités matérielles ne sont pas identiques, les interactions changent et les contraintes aussi. En revanche, au moins, sur les parties desktop et mobile, on peut rapprocher les architectures et unifier les modèles applicatifs.

Windows Polaris

Récemment, une rumeur autour d'un projet Windows a fait parler d'elle : Windows Polaris. Ce projet serait une refonte de l'architecture de Windows plutôt orientée desktop, ordinateurs portables. Un des fondements sera Windows Core OS, une fondation unique du système permettant d'être modulaire et d'adapter le système au matériel cible et en adoptant uniquement les applications universelles UWAP. La rumeur évoquée par Windows Central évoque une possible sortie courant 2019… ou pas. Ce n'est pas la première fois que l'on entend parler d'un Windows modulaire. Un des projets connus était Singularity.

Un problème de cycle ?

Le cycle de développement et donc de mise à disposition d'une nouvelle version d'OS est au coeur des questions actuelles. Il est déjà difficile de gérer les cycles des applications entre les mises à jour mineures pour corriger des bugs gênants ou stabilité des fonctionnalités et les versions majeures rajoutant de nouvelles fonctions, imaginez ce que cela donne avec un OS.
Faut-il s'obstiner à vouloir à tout prix sortir une "grosse" version par an sachant que de nombreuses nouveautés sont souvent mineures ou à intérêt réduit ? L'avantage est ne pas attendre 2-3 ans, voire plus, entre deux versions. Pour l'utilisateur, c'est l'impression d'une évolution suivie et régulière, même s'il ne comprend pas toujours l'intérêt de la nouvelle itération. Mais pour les équipes de développement, la pression est constante et bien plus forte même si on lisse les nouveautés et améliorations, quitte à décaler d'une version. Apple avait adopté cette cadence, Microsoft et Google ont suivi. Côté Linux, c'est le cas aussi.

Depuis octobre dernier, les OS d'Apple, iOS 11 et macOS High Sierra, ont cumulé les failles, les bugs, les problèmes, les retards de fonctionnalités et… les mises à jour. Chaque année, nous avons le même rituel : version majeure puis 1-2 mois après, une première grosse mise à jour pour corriger les principaux problèmes. Mais cette année, tout est parti de travers. Et plusieurs fonctions attendues depuis des mois comme les évolutions de iMessage ou d'AirPlay ont patiné. Ceci tout en menant en parallèle de gros travaux de rénovation dans le coeur du système : BridgeOS (OS embarqué dérivant de watchOS pour gérer les puces ARM des Mac), APFS (nouveau système de fichiers), le support des e-GPU, la fin des apps 32 bits, etc. Et visiblement, Apple a décidé de faire un travail d'optimisation et de stabilité pour cette année sur macOS et surtout iOS.

Mais le problème existe aussi pour Windows, Android et Linux.

La pression des utilisateurs qui en veulent toujours plus, même si au final, on n'utilise qu'un nombre restreint de fonctions systèmes, ne facilite pas le travail des équipes. Mais finalement, n'est-ce pas aux éditeurs de prendre leurs responsabilités et de dire stop ? Ne sommes-nous pas à la fin d'un cycle ? Ne peut-on pas imaginer une informatique sans OS ? •

Daniel DJORDJEVIC
Développeur chez Infinite Square
https://blogs.infinitesquare.com/
Sébastien OLLIVIER
Développeur Web Full Stack
https://sebastienollivier.fr/blog

Angular 5

Angular 5, la nouvelle version du framework, nommée Pentagonal-donut, est sortie le 1er décembre 2017. Cette version a pour objectif d'améliorer les performances des applications Angular ainsi que d'optimiser tout ce qui tourne autour de la compilation.

DE MEILLEURES PERFORMANCES

Angular CLI et Build Optimizer

Conjointement à Angular 5, la CLI est aussi mise à jour. Angular CLI 1.5 utilise maintenant le Build Optimizer par défaut.
Ce Build Optimizer réalise plusieurs optimisations supplémentaires, qui permettent notamment un meilleur tree shaking (https://developer.mozilla.org/en-US/docs/Glossary/Tree_shaking).
Le compilateur Angular est bien entendu lui aussi amélioré/optimisé, ce qui permet (sans surprise) des compilations plus rapides. Une des améliorations très intéressantes pour le développeur est le support de la compilation incrémentale par le compilateur Angular. Cela est possible, car le compilateur Angular opère comme un TypeScript transform (disponible depuis Typescript 2.3), ce qui permet au compilateur de se brancher sur le pipeline de compilation TypeScript.
Concrètement, cela permet au développeur de travailler en AOT, sans avoir des temps de compilation indécents.

```
1 ng serve --aot
```

Nous bénéficions alors de tous les avantages de la build AOT, avec la validation des templates notamment. (voir Angular : JIT vs AOT)
Les développeurs d'Angular veulent amener la CLI vers ce comportement par défaut, car leur objectif est de rendre la compilation AOT assez rapide pour que les développeurs n'aient pas à développer en JIT. Cela nous évitera les mauvaises surprises lorsqu'on génère un package de production et qu'on découvre un écart avec la version JIT.
Concrètement, la build AOT du site https://angular.io est passée de plus de 40 secondes de build à 2 secondes, grâce à la compilation incrémentale.

Preserve Whitespaces

Avant cette nouvelle version, le compilateur préservait les espaces, les sauts de lignes et les tabulations présentes dans les templates de composants. Maintenant, il est possible de choisir si l'on veut les préserver ou non, sachant que la feature est désactivée par défaut.
Selon les spécifications, l'implémentation actuelle se comporte de la sorte :

- Tous les whitespaces au début et à la fin du template sont retirés (trim)
- Les nœuds de texte ne contenant que des whitespaces seront retirés

```
Avant : <div>contenu 1</div> <div>contenu 2</div>
Après : <div>contenu 1</div><div>contenu 2</div>
```

- Une série de whitespaces consécutifs dans un nœud de texte sera remplacée par un espace unique.

```
2 Avant : <div>\n mon contenu \n</div>
Après : <div> mon contenu </div>
```

- Le contenu des balises dont les whitespace ont de l'importance n'est pas modifié (balise <pre> par exemple)
- Il est possible de forcer un espace obligatoirement avec &ngsp;

Cette configuration peut se faire à deux endroits.
Soit au niveau du composant directement, dans le décorateur @Component.

```
1 @Component({
2    templateUrl: 'no-whitespaces.component.html',
3    preserveWhitespaces: false
4 })
5 export class NoWhiteSpacesComponent
```

Soit de manière globale, pour toute l'application, au niveau du fichier tsconfig.json.

```
1 {
2    "compilerOptions": {...},
3    "angularCompilerOptions": {
4       "preserveWhitespaces": false
5    },
6    "exclude": {...}
7 }
```

RxJS 5.5

La dépendance vers RxJS est enfin mise à jour : c'est la version 5.5.2+ qui est maintenant utilisée, qui supporte les modules ECMAScript, ce qui signifie que le tree shaking permet un bundle final encore plus léger. Historiquement, d'un point de vue développeur, il fallait faire attention en important les opérateurs d'Observable, sans quoi on se retrouvait avec tous les opérateurs dans le bundle final, tandis que nous en utilisons seulement deux.
Il fallait alors importer les opérateurs de la sorte :

```
1 import 'rxjs/add/operator/map';
2 import 'rxjs/add/operator/finally';
```

Maintenant, il suffit d'écrire :

```
1 import { map, filter } from 'rxjs/operators';
```

UPDATEON BLUR/SUBMIT AVEC ANGULAR FORMS

Il est maintenant possible de lancer une validation ainsi qu'une mise à jour des données sur le blur et le submit, et non plus sur tous les évènements d'input.
Pour cela, il suffit de renseigner le paramètre updateOn :

Version Template Driven Forms

```
1 <!-- Sur le formulaire-->
2 <form [ngFormOptions]="{updateOn: 'submit'}">
3
4 <!-- Sur un contrôle spécifique -->
5 <input name="myInput" ngModel [ngModelOptions]="{updateOn: 'blur'}"
```

Version Reactive Forms

```
1 // Sur un groupe
2 new FormGroup(value, {updateOn: 'blur'}));
3
4 // Sur un contrôle
5 new FormControl(value, {updateOn:'blur'}));
```

DE NOUVEAUX ÉVÈNEMENTS SUR LE CYCLE DE VIE DU ROUTEUR

De nouveaux évènements sont mis à disposition pour le cycle de vie du routeur. L'objectif des équipes Angular est de permettre aux développeurs de tester les performances des guards, ou bien de pouvoir gérer le statut d'un loader sur un router-outlet spécifique.
Les nouveaux évènements sont les suivants, leur nom étant assez explicite :

- GuardsCheckStart
- ChildActivationStart
- ActivationStart
- GuardsCheckEnd
- ResolveStart
- ResolveEnd
- ActivationEnd
- ChildActivationEnd

Pour gérer un loader sur une activation de route enfant, on pourrait alors faire (extrait du blog angular) :

```
1 class MyComponent {
2   constructor(public router: Router, spinner: Spinner) {
3     router.events.subscribe(e => {
4       if (e instanceof ChildActivationStart) {
5         spinner.start(e.route);
6       } else if (e instanceof ChildActivationEnd) {
7         spinner.end(e.route);
8       }
9     });
10 }
11 }
```

LES PIPES INTERNATIONALISÉS : NUMBER, DATE ET CURRENCY

Historiquement, les pipes cités ont été développés en utilisant les API des navigateurs pour récupérer les formats de nombres, dates et les devises. Il fallait alors ajouter des polyfills (des scripts supplémentaires pour rajouter des fonctionnalités manquantes aux API de certains navigateurs), et le résultat n'était quand même pas satisfaisant.
C'est pourquoi les pipes ont été refaits, et utilisent uniquement une implémentation propre aux équipes d'Angular, se basant sur le Unicode CLDR, ce qui apporte un meilleur support des locales et une configuration beaucoup plus aisée.
Si, pour une raison quelconque, le développeur souhaite utiliser les anciens pipes, il faut désormais importer le module DeprecatedI18NPipesModule.

EN CONCLUSION

Comme nous l'avons vu, cette version n'apporte pas de gros changements au niveau API, mais l'équipe d'Angular continue d'optimiser son Framework, pour qu'il soit toujours plus léger, plus performant et mieux outillé (on le voit notamment avec les efforts fournis au niveau de la CLI et de la build).
Une version 6, qui devrait être en release fin mars début avril, a déjà été annoncée, de même qu'une future version 7, fin 2018, et 8, début 2019, ce qui montre l'ambition de l'équipe de Google à délivrer un framework toujours plus abouti.
Un bel avenir en perspective… •

Michaël Bertocchi
Ingénieur développement
@dupot_org

Développer pour le mobile avec une technologie multiplateforme

PARTIE 1: ÉTAT DES LIEUX

Lorsque vous devez développer une application sur les deux principales plateformes mobiles iOS et Android (je pense que l'on peut désormais oublier Windows Phone), vous avez plusieurs solutions disponibles : framework hybride (Cordova / Phonegap), Qt framework, Xamarin, framework JavaScript "native" (React Native/Nativescript).

Une parenthèse sur les frameworks "natives" (dont j'ai entendu parler très récemment) : l'application est écrite en JavaScript + un metalangage ressemblant à de l'html, mais contrairement à Cordova/Phonegap, le framework transcrit en code natif (pas d'utilisation de webview), ça peut faire penser à Haxe, pour ceux qui connaissent. Ne connaissant ni Xamarin, ni les frameworks "natives", j'aborderai ici le cas des deux premiers. Je présenterai d'abord chaque technologie, décrirait les avantages/inconvénients et enfin présenterai les outils de développements associés.

CORDOVA / PHONEGAP

Présentation

Le framework propose de programmer avec les technologies web (JavaScript et HTML/CSS) un mini site puis de générer une application composée d'un moteur de rendu web en plein écran (webview) pour charger celui-ci. Et pour augmenter les capacités de vos applications, le framework permet de connecter des éléments non accessibles en web via un système d'api (téléphone, sms, appareil photo, géolocalisation, gyroscope…).

Les outils de développement

Utilisant des technologies web, un simple éditeur de texte permettra de commencer (notepadd++, KDevelop, Smultron…). Vous pourrez bien sûr vous orienter vers des IDE plus poussés : la plupart proposant des prises en charge de Cordova/Phonegap (Atom, Netbeans, SublimeText…).

Pour utiliser ce framework, il vous faut installer nodeJS, puis npm (gestionnaire de paquets) pour enfin installer l'un des deux frameworks : npm install -g cordova ou npm install -g phonegap

En vous rendant dans le répertoire de votre projet, vous pouvez lancer une instance serveur écoutant sur le port 3000. Via la commande : phonegap serve ou cordova serve

A partir de là, vous aurez le choix :
soit d'ouvrir un navigateur web sur votre adresse locale (http://localhost:3000) ;
soit, d'installer l'application mobile pour les développeurs (« Phonegap Developer » (disponible sur Android et iOS) et d'ouvrir votre instance directement depuis votre mobile.
Vous pouvez donc ici tester sans avoir besoin de connecter un câble usb ou de compiler votre projet. C'est très pratique pour tester rapidement sur plusieurs plateformes différentes (mobiles, tablettes Android/iOs) et ceci sans Mac (ni l'installation des différents SDK habituels) 1

La communauté

L'histoire de Phonegap/Cordova a créé deux produits distincts partageant beaucoup. Vous pouvez chercher de l'aide autant sur le nom Phonegap que Cordova :
- directement sur le site d'Adobe : https://forums.adobe.com/community/phonegap ;
- sur les sites habituels de développeurs : developpez.com et openclassrooms.com pour les francophones ;
- ou l'inconditionnel stackoverflow.com.

De plus, utilisant les technologies web, vous pouvez vous faire aider par les communautés historiques de création web. En effet, la majorité du temps vous aurez plus des soucis, interrogations concernant du JavaScript / HTML/CSS que du framework lui-même.
Car mis à part les ponts pour accéder aux éléments du téléphone (sms, gyroscope, caméra…), le gros de l'application restera le développement de l'interface et l'écriture d'algorithmes pour manipuler, présenter ou consommer vos données.

1

La documentation

Comme pour la communauté, vous pouvez autant trouver votre bonheur sur le site d'Adobe que sur celui de Cordova :
Apache Cordova: https://cordova.apache.org/ ;
Adobe Phonegap : http://docs.phonegap.com/ ;
JavaScript/HTML/CSS : https://www.w3schools.com/ .

Prix / Licensing

Apache Cordova et Adobe Phonegap sont des framework open source, gratuits à utiliser même pour une utilisation commerciale. Le prix des IDE, lui, dépendra de vos propres choix, car il n'y en a pas d'officiel.
L'application Android/iOS permettant de tester son application sur son mobile est également gratuite.

Note

Pour ceux qui n'aiment pas la ligne de commande, Adobe propose pour Phonegap une application en beta pour Windows et macOS : « Phoneg Desktop App »

La seule partie payante, si elle vous intéresse, est le service de compilation sur le Cloud: « Phonegap Build » https://build.phonegap.com/ .
En effet celui-ci propose divers tarif : de gratuit pour les projets opensource à 9,99 $ par mois pour les projets privés.
Plus de détails ici : https://build.phonegap.com/

Les avantages

Premièrement le choix des langages utilisés permet un accès facilité à beaucoup de développeurs confirmés ainsi qu'à de nombreux débutants. En effet, le JavaScript et l'HTML/CSS ont une courbe d'apprentissage légère et sont connus d'un très grand nombre de développeurs aujourd'hui.
De plus, beaucoup de sites proposant déjà une version adaptée aux mobiles (responsive, adaptative), on ne part pas de zéro pour concevoir l'application.
Mais attention, si certains sites utilisent un frontend riche, usant de framework JavaScript tel que React, Angular ou Aurelia, d'autres sont plus « traditionnels » et mélangent le l'HTML/CSS + JavaScript coté client et du code serveur (php, Java,C#…)
Attention, dans cette optique de migrer le site en une application mobile, Il faudra faire un exercice de simplification/allègement du code : en effet il ne faut jamais oublier qu'un mobile n'est pas un ordinateur : son processeur est moins performant, et sa batterie étant un élément très critique pour son propriétaire, il faut veiller à l'économiser au maximum.

Deuxième avantage, le temps de développement : n'ayant pas de phase de compilation pour tester, un simple enregistrement du fichier suffit à rafraîchir l'application quasi instantanément.

Troisième avantage: la légèreté de l'environnement de développement couplée à l'application mobile de visualisation du résultat directement sur son mobile permettent de travailler avec un ordinateur « entrée de gamme » (un simple Intel Atom, ou un ARM comme sur les Raspberry Pi/Chromebooks permettent de développer vos applications).

Quatrième et dernier avantage « last but not least »: Adobe propose un service de cross compilation sur le Cloud: ainsi vous connecter votre compte Github et eux se charge de la compilation Android, iOS… https://build.phonegap.com

Les inconvénients

Il y a deux inconvénients que l'on peut faire à cette technologie: les performances et le rendu web. Pour les performances, après de nombreuses améliorations, ce point reste à nuancer, mais il faudra quand même le garder en tête.
Cette technologie interprétant une application web, vous avez donc les mêmes contraintes de rapidité qu'un site web. Éviter de faire des jeux ou des applications gourmandes en calcul avec cette technologie. Je ne dis pas ici que vous ne pouvez pas écrire des jeux, mais si c'est le cas, ils devront être peu gourmands en animations, nombre d'éléments à gérer à l'écran.
Vous pourriez par exemple faire un jeu de plateforme, de plateau, une application de VTC…
Concernant le rendu web, il faudra user de CSS, JavaScript pour avoir un rendu le plus proche possible d'une application native que d'un site web.
Certains frameworks proposent des objets simulant des éléments natifs, mais à force d'ajouter des frameworks CSS/JavaScript, etc., vous alourdirez l'application finale et en paierez le prix à l'usage : soit en termes de performance, soit en termes de consommation d'énergie. Quelquefois « le mieux est l'ennemi du bien ».

QT FRAMEWORK

Présentation

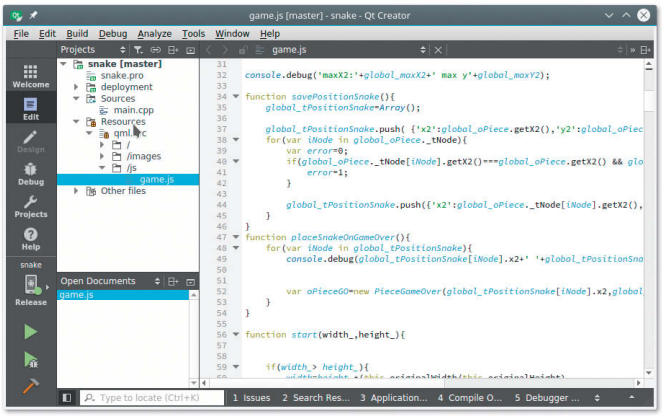
L'approche ici est différente : plutôt que d'encapsuler du web dans un navigateur plein écran, l'idée ici est d'écrire et de compiler du code C++ pour ensuite l'intégrer à une application native générée pour l'occasion.
Les technologies utilisées sont le C++, le JavaScript et le QML. L'alliance de ces trois langages, renforcé par la présence de librairies/api performantes, permettent de pouvoir développer des applications de qualité et performantes.
A la base vous créez une classe C++, celle-ci instancie des objets d'affichage écrits en QML (syntaxe ressemblant à du CSS), et vous pouvez utiliser du JavaScript pour interagir avec les objets QML mais également avec le C++.

Les outils de développement

Soit vous développez avec votre IDE C++ habituel, soit vous utilisez l'éditeur spécialement conçu pour : Qt Creator [figure 2].
Ce logiciel multiplateforme, gère les trois langages du framework (C++, JavaScript et QML), et comme tout IDE digne de ce nom propose l'auto-complétion contextuelle (même entre les trois couches de langage), le débogage, la création graphique d'interface…
Le framework permet de compiler vers des plateformes mobiles mais également vers des OS d'ordinateur. Vous pouvez donc compiler pour votre OS local, ceci permet très rapidement d'avoir un rendu de l'application et de pouvoir ainsi avancer dans sa création plus rapidement.
L'outil se veut assez léger à démarrer et utiliser, certes plus lourd qu'un notepad++, vous aurez l'agréable sentiment de légèreté/réactivité en l'utilisant.
Conçu spécialement par la communauté pour développer avec ce framework, on bénéficie du confort d'utilisation et de la simplification de l'interface. Il est facile à prendre en main, à utiliser et se retrouver dans l'arborescence de son projet.
De plus il intègre à la fois les habituels outils pour déboguer, compiler, exécuter, mais également des interfaces pour paramétrer les fichiers de destination comme les fichiers propres à Android.
Vous pourrez ainsi aussi bien paramétrer les informations de votre application mobile, ses différents logos ainsi que votre clé de signature. Clé nécessaire pour signer votre paquet d'installation apk avant de le soumettre au play store Android. 2

La communauté

Qt n'est pas un framework comme un autre, il a la chance d'avoir été choisi depuis des années pour développer un des principaux environnements de bureau les plus utilisées sous GNU/Linux : KDE
Ce faisant, (pan pan) il est utilisé pour développer de nombreuses applications dont KDevelopp, Dolphin, Amarok… vous avez donc la possibilité de solliciter les communautés Qt, KDE et GNU/Linux
Vous pouvez trouver de l'aide soit :
sur le site de qt https://forum.qt.io/ ;
sur les sites de developpez.com
https://qt.developpez.com/ ;



2

sur la communauté GNU/Linux https://askubuntu.com ; KDE qui vous redirige sur Qtcentre http://www.qtcentre.org .
Utilisant également des langages plus standards comme le C++ et le JavaScript, vous pourrez également faire appel à des communautés plus traditionnelles d'entraide sur ces langages (openclassrooms.com).

Note

Le JavaScript utilisé ici est un peu différent de celui utilisé dans vos applications web, n'essayez donc pas d'appeler le DOM pour récupérer un élément via son ID.

La documentation

La documentation est assez fournie sur le site officiel du framework : Site de Qt : http://doc.qt.io/ ;
De plus l'outil de développement, Qt Creator permet de charger une liste d'applications d'exemples. Ceci permet à la fois de les compiler pour voir le rendu mais également d'analyser le code source pour mieux comprendre.

Prix / Licensing

Qt propose deux licences : gratuit pour les projets opensource et 459$ par mois pour les produits commerciaux.
La licence commerciale incluant bien plus que la possibilité de commercialiser vos applications : un support ainsi que des fonctionnalités supplémentaires.
Plus de détails sur cette page :
https://www.qt.io/download
Il en est de même pour l'IDE officiel : gratuit pour des projets opensource et payant pour les produits commerciaux (inclus dans la licence commerciale de Qt).

Type	Temps d'apprentissage	Performance	Type d'application
Hybride (Cordova/Phonegap)	Court	Moyenne	Jeu peu gourmand, applications légères, version mobiles de site web
Framework Qt	Moyen	Très bonne	Tout type de jeux, applications gourmandes
Natif (Android/iOS)	Long	Bonne	Tous type d'applications(*)

(*) Note : pour les jeux, le site officiel d'Android propose d'utiliser NDK (utilisé par Qt) pour les applications gourmandes.

Avantages

Le premier avantage, surtout comparé à la précédente solution c'est justement les performances : avec ce framework vous avez, malgré le côté multiplateforme, des performances dignes d'une application native, voir plus rapide (C++ compilé étant plus rapide que du Java précompilé).

Deuxièmement, l'homogénéité graphique: l'application étant compilée avec ce framework, vous utilisez des éléments d'interface du framework avec un rendu similaire, peu importe la plateforme cible (même affichage sur votre ordinateur, votre Android ou votre mobile).

Troisièmement, le réel coté multiplateforme: comme la fameuse guideline de Java "write once, run anywhere", ici une application développée peut être compilée vers Android, iOS et même Ordinateur (Windows, Mac et GNU/Linux).

Quatrièmement, la flexibilité des langages d'écriture possible : proposant à la fois C++ et JavaScript pour le code de l'application, vous avez la possibilité, soit :

- De mixer les trois langages avec leurs forces respectives ;
- D'utiliser uniquement C++ et QML ;
- Ou développer toute l'application en utilisant uniquement JavaScript et QML*.

Inconvénients

Le seul reproche que l'on puisse faire à cette technologie serait le choix du C++ pour ceux qui n'aiment pas ce type de langage. Mais, comme indiqué précédemment, les équipes ont ajouté la possibilité de pouvoir écrire une partie (voir la totalité*) de son application uniquement en utilisant du JavaScript et du QML.

* *Une classe Main C++ servant uniquement à instancier l'application.*

Comparatif

Extrait du site d'Android :
« Squeeze extra performance out of a device to achieve low latency or run computationally intensive applications, such as games or physics simulations »
https://developer.android.com/ndk/guides/index.html

Lequel choisir ?

Premièrement, il faut se demander si vous faites ou non une application open source ; en fonction de cette première question, les prix de ces deux solutions va fortement faire la différence.
Deuxième élément, est-ce une application gourmande ou non ? Ce second point peut également faire pencher sensiblement la balance. Si celle-ci ne nécessite pas de performances élevées: un petit jeu en 2D, une application de contenu légère (comme un journal de bord, une application de CRM..), vous aurez réellement le choix des armes selon vos préférences personnelles.
En revanche pour les applications plus gourmandes il vous faudra vous tourner vers Qt et Xamarin.
Troisième élément : si vous souhaitez déployer sur la plateforme à la pomme, disposez-vous d'un Mac ? Question qui peut sembler bête, mais on ne peut compiler pour iOS qu'avec un Mac ; c'est ici un point qui peut, grâce à sa plateforme Cloud, faire pencher la balance vers la solution Hybride d'Adobe.

Conclusion

Ces deux technologies ont leurs avantages et inconvénients, mais ce qui importe surtout c'est de savoir quel besoin de performance vous avez.
Je vous invite néanmoins à tester ces technologies pour vous faire votre propre avis, et même si vous choisissez d'opter pour une solution multiplateforme « compilée » (comme Xamarin ou Qt), il est fort pertinent de tester au moins une fois Phonegap/Cordova : cette technologie pourrait répondre à vos besoins ultérieurement.
Par exemple, en ce moment je dois faire une application de journal de bord multiplateforme pour le sport :

- C'est un projet open source (comme tout ce que je fais) ;
- C'est une application légère (un journal de bord pour suivre son évolution) ;
- Elle doit être disponible sur Android et iOS ;
- Je n'ai pas de Mac pour compiler.

Le choix va de soi : j'ai commencé à développer sur Phonegap et j'utiliserai sans doute leur service de compilation Cloud pour déployer sur le Play store et l'App store et ceci, sans Mac :). •

Christophe Gigax
MVP Microsoft
Visual Studio & Development Technologies

Visual Studio Code : l'éditeur Open Source de Microsoft

Poussé par Satya Nadella, Microsoft se tourne désormais vers l'Open Source, et chaque parcelle de l'écosystème ne déroge pas à la règle. D'un point de vue technologique, on a d'abord vu plusieurs bibliothèques et compilateurs clés être publiés sur GitHub. Ensuite, les équipes ont travaillé sur une version Open Source et multiplateforme de l'IDE ultrapopulaire : Visual Studio. Intitulé Visual Studio Code, cet éditeur s'inspire grandement de Atom, tout en proposant une expérience enrichissante à tous les niveaux : langage de développement, debug, contrôleur de code source … Qu'apporte ce nouvel éditeur par rapport à ses concurrents ? Est-il vraiment l'éditeur adapté aux développements de demain ?

L'open-source jusqu'au bout

La grande nouveauté de VS Code est son appartenance au monde Open Source : https://github.com/Microsoft/vscode. Selon une étude publiée par GitHub il y a peu de temps, VS Code est le projet avec le plus de contributeur (15 000 contributeurs) et parmi les 10 repositories les plus discutés sur la plateforme (l'étude : https://octoverse.github.com/). 1

Cette première position est tout de même à nuancer, car on parle des contributeurs de Visual Studio Code avec les extensions de l'éditeur. Cependant, cela montre tout de même le ralliement de la communauté à ce projet faisant aujourd'hui de VS Code l'un des éditeurs Open Source les plus utilisés au monde. Il se présente sous la forme d'une application Web développée avec Electron. Cette technologie permet de transformer une application Web en application Desktop multiplateforme via Node.js. C'est notamment cette partie de l'éditeur qui fait sa force : il est installable sur Windows mais aussi sur Linux et macOS : l'impact auprès de la communauté et ainsi garanti et amplifié.

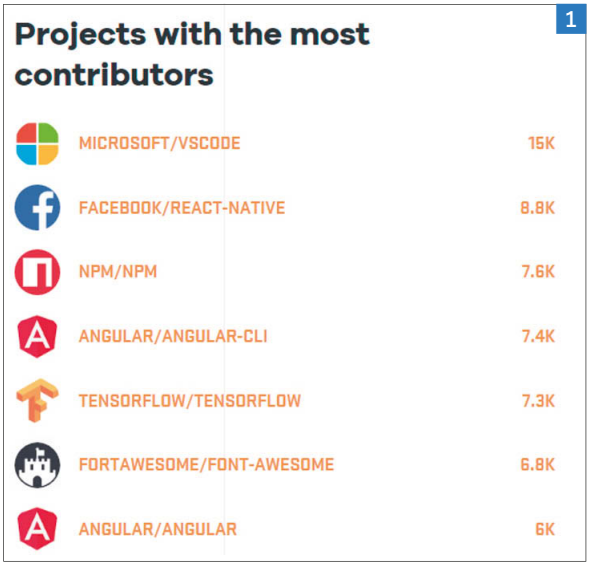
Grâce au travail de toute cette communauté grandissante, l'éditeur est mis à jour au moins une fois par mois, prouvant ainsi que l'outil est amené à évoluer avec son temps tout en garantissant un haut niveau de fiabilité. C'est également la force de l'éditeur : il est très performant et offre des fonctionnalités réactives afin de conserver une expérience utilisateur fluide. Il est notamment possible de contrôler les performances de l'éditeur via la commande suivante :

```
> code –status
```

Cette commande permet de lister plusieurs choses :

- Les processus de VS Code en cours d'exécution ;
- Les tâches que l'éditeur est en train de faire fonctionner par processus VS Code ouvert ;
- Les extensions en cours d'exécution ;
- Les terminaux et autres fenêtres ouverts (recherche, debug …) ;
- Les états des workspaces ouverts.
- Le développeur peut alors diagnostiquer la lenteur de son VS Code et essayant plusieurs combinaisons de l'éditeur. Par exemple, il peut lancer son éditeur avec la commande suivante :

```
> code –disable-extensions
```



Liste des projets avec le plus de contributeurs sur GitHub

Cette option permet de lancer VS Code sans extension, et ainsi valider que le problème vient d'une extension ou non. Si tel est le cas, le développeur peut générer un profile CPU via la commande *Developer: Show Running Extensions* (à lancer dans VS Code) afin de l'envoyer lors de la remontée d'erreur.

Au final, le maître mot de l'éditeur est la transparence et la simplicité tout en proposant des fonctionnalités avancées (IntelliSense, coloration syntaxique …) boostant la productivité du développeur.

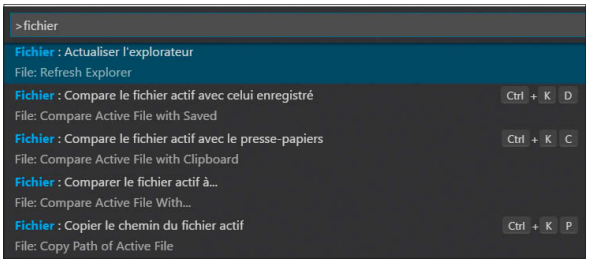
Un éditeur simple d'utilisation et multi langage

La simplicité de l'éditeur passe tout d'abord par une interface graphique épurée. On ne retrouve pas plus que :

- Une barre d'outils à gauche pour un accès rapide aux grandes fonctionnalités de l'éditeur ;
- Un explorateur de fichiers à côté ;
- Une barre de menu en haut ;
- Des fenêtres ancrées en bas type console, erreur ou terminal ;
- L'éditeur au milieu.

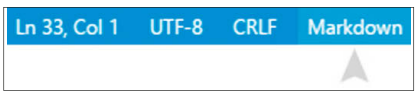
Malgré sa discrétion, la barre de menu offre une multitude de com-

mandes où chacune possède son raccourci clavier.
Le développeur peut ainsi rapidement utiliser certaines fonctions depuis son clavier. 2
Visual Studio Code propose également un exécuteur de commandes intégré et facilement accessible avec le raccourci **Ctrl+Maj+P**. Les commandes permettent de piloter VS Code uniquement avec le clavier. Cet utilitaire est très pratique lorsqu'on a l'habitude de travailler principalement au clavier. Un rappel des raccourcies apparaît à droite de la liste. La liste est triée en fonction de ce que le développeur écrit.



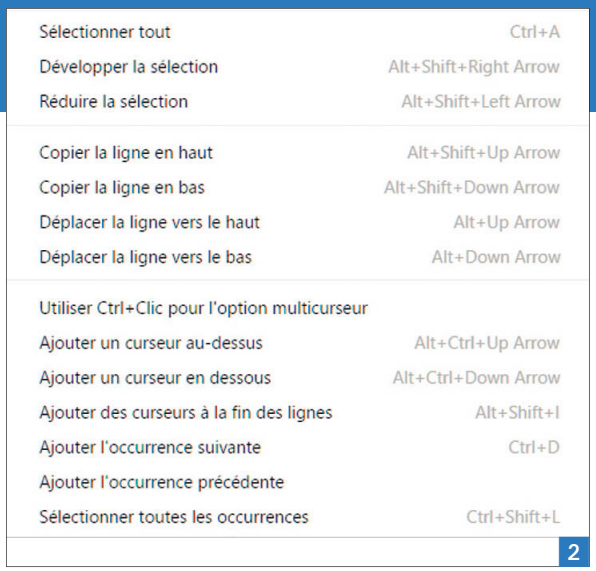
3 - *Palette de commande pour gérer les fichiers*

L'éditeur fait également beaucoup d'efforts en termes de navigation puisqu'il propose plusieurs raccourcis pour passer, par exemple, d'un fichier à un autre ou d'un onglet à l'autre.
Visual Studio Code supporte la majorité des langages des programmations modernes tel que C++, C#, Java, JavaScript, TypeScript et plus encore. Si jamais votre langage n'est pas supporté, il suffit de chercher le plugin qui le fera supporter sur le VS Code Marketplace. Pour .NET spécifiquement, VS Code utilise toute la puissance du projet OmniSharp. Ce projet vise à apporter .NET sur toutes les plateformes comme Windows, Linux ou encore Mac et surtout sur n'importe quel éditeur : Atom, Sublime Text ou même Vim ! Le principe est simple : OmniSharp fonctionne comme un serveur HTTP et embarque un analyseur syntaxique (Roselyn en l'occurrence) C#. Lorsque VS Code (ou votre éditeur préféré avec l'extension installée) a besoin d'IntelliSense pour .NET, il lance une requête HTTP en asynchrone et interroge le serveur OmniSharp (qui tourne comme un process normal en tâche de fond). Ce dernier lui répond et l'éditeur peut afficher le résultat sous la forme de l'IntelliSense. Cette architecture permet notamment de conserver un corps commun de traitement et autonome, où plusieurs plugins selon l'éditeur viennent requêter.
Visual Studio Code se base sur l'extension du fichier pour déterminer le langage à utiliser. Cependant, il est possible de changer de mode de langage si cela s'avère nécessaire. Pour ce faire, il suffit de cliquer sur l'éditeur de langue situé en bas à droite de la fenêtre.



4 - *Sélection d'un langage via la barre d'outils*

Chaque langage se voit associer un **Language Id**. Cet identifiant sert à associer certaines fonctions de l'éditeur à des langages bien spécifiques (auto-import, auto-complétions, prévisualisation …). Cet identifiant est une chaîne de caractères souvent (pas toujours)



Les raccourcis clavier pour chaque commande de VS Code

en minuscules. Par exemple, pour Batch c'est **bat**, ou C# c'est **csharp**. Il est ensuite possible d'étendre les fonctionnalités liées aux langages à d'autres extensions de fichiers qui ne sont pas connus d'origine via un paramétrage intitulé *files.associations*. L'exemple ci-dessous associe le langage PHP aux extensions **.myphp**.

```
"files.associations": {
"*.myphp": "php"
}
```

Un espace de travail léger et personnalisé

Visual Studio Code met un point d'honneur à la personnalisation de l'espace de travail du développeur. Il peut ainsi paramétrer un espace de travail par utilisateur, ou alors carrément configurer plusieurs espaces de travail par utilisateur, selon les projets. Ces paramètres sont stockés dans un dossier **.vscode**, et ne s'activent que lorsque l'espace de travail est ouvert dans l'éditeur. Cela peut être pratique si vous désirez partager des paramètres communs pour toute une équipe.
Les paramètres de l'espace de travail sont représentés sous la forme d'un fichier JSON intitulé **settings.json**. L'éditeur propose même une auto-complétions suivant le paramétrage désiré afin d'aider la saisie.

- Sous **Windows** : Fichier > Préférences > Paramètres ;
- Sous **Mac** : Code > Préférences > Paramètres.

Ce fichier de paramétrage est stocké aux endroits suivants :

- Sous **Windows** : %APPDATA%\Code\User\settings.json ;
- Sous **Mac** : $HOME/Library/Application Support/Code/User/settings.json
- Sous **Linux** : $HOME/.config/Code/User/settings.json

A gauche, on retrouve les paramètres par défaut qui sont appliqués par l'éditeur. A droite, ce sont les paramètres par utilisateur. Il suffit alors de copier / coller les paramètres pour les personnaliser facilement. En haut de la fenêtre se trouve une barre de recherche. Elle permet de filtrer selon la saisie, puis propose des actions rapides pour faciliter le dédoublement. Chaque nouveau plugin peut potentiellement rajouter des paramètres dans les paramétrages par défaut : ils apparaîtront également dans le fichier de gauche. De plus, pour faciliter la navigation, les paramètres sont regroupés sous un même nom.

Le debug sans concession

L'une des grandes fonctionnalités de VS Code est le debug. De base, l'éditeur permet de débugger les applications NodeJS écrites en JavaScript, TypeScript ou tout autre langage transpilable en JavaScript. Concernant les autres langages, il faut installer des plugins qui vont permettre de débugger dans d'autres langages comme Python, C++, C# … Ces plugins se trouvent dans le VS Code Marketplace. 5

L'onglet debug propose une disposition de l'information très simple. On retrouve :

- **Sur la gauche** : les informations de debug telles que la pile des appels, les variables locales / globales, les points d'arrêt et les variables espionnées ;
- **En bas** : la console de debug ;
- **En haut à gauche** : les commandes de lancement et de configuration des paramètres de debug car il est possible en effet de configurer plusieurs profils de debug selon l'application que l'on veut debugger.
- **En haut à droite** : les commandes d'utilisation du debug : pas-à-pas, pause, arrêt …

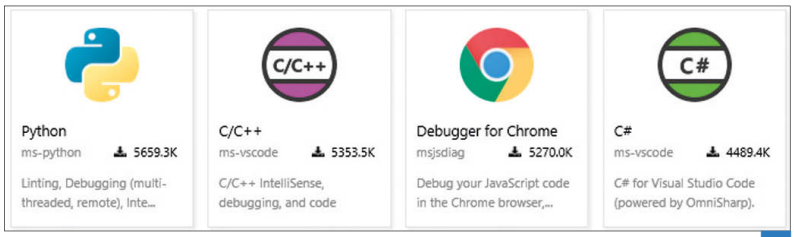
Le menu debug récapitule toutes les commandes possibles pour le debug. De base, sans paramétrage supplémentaire, VS Code va lancer le debug du fichier courant comme si c'était une application NodeJS. Pour des scénarios plus complexes, il suffit de créer des configurations de debug via le fichier **launch.json**. Ce fichier se crée automatiquement et VS Code va tenter de générer une configuration préalable selon l'environnement courant ouvert (application NodeJS, Angular, ASP.NET Core …). Pour une application .NET Core, on peut trouver une configuration de ce type :

```
{
  "version": "0.2.0",
  "configurations": [
   {
    "name": ".NET Core Launch (console)",
    "type": "coreclr",
    "request": "launch",
    "preLaunchTask": "build",
    "program": "${workspaceRoot}/myApp.dll",
    "args": [],
    "cwd": "${workspaceRoot}"
   }
  ]
}
```

Chaque propriété peut être différente en fonction de la propriété « type » indiquée. Pour rajouter rapidement une configuration, il suffit de cliquer sur le bouton en bas à droite. L'IntelliSense de VS Code aide ensuite le développeur à trouver le bon snippet, puis les bonnes propriétés.

Les principales propriétés sont les suivantes :

- **type** : définit quel debugger sera utilisé. Certains plugins vont définir leur propre valeur à cet endroit. On va par exemple retrouver **php** pour le compilateur PHP introduit par l'extension prévue à cet effet ;
- **request** : type de debug. Cela peut être de type **launch** si le développeur souhaite lancer l'application, sinon **attach** s'il souhaite s'attacher à un processus en cours ;
- **name** : nom qui va apparaître dans la liste déroulante.



Différents plugins pour débugger d'autres langages 5

Certains debugger rajoutent leurs propres attributs comme :

- **program** : programme à exécuter pour le lancement du debug ;
- **args** : arguments à donner ;
- **env** : variable d'environnement ;
- **port** : port à utiliser.

Les possibilités offertes par cette fonctionnalité de l'IDE sont énormes, et chaque plugin va pouvoir agrémenter sa configuration comme il le souhaite. De plus, le développeur peut utiliser plusieurs variables dans ce fichier de configuration comme par exemple :

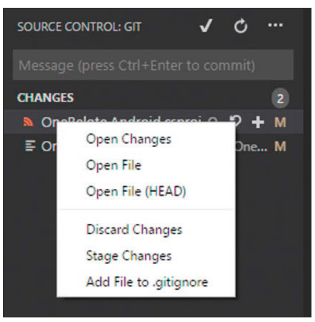
- **${workspaceFolder}** : chemin vers l'espace de travail courant ;
- **${file}** : fichier courant ouvert ;
- **${fileBasename}** : nom du fichier courant ;
- **${lineNumber}** : numéro de ligne actuellement sélectionnée.

Ainsi, de manière très extensible, le développeur peut agir sur les paramètres de son debug depuis son environnement de travail dans l'IDE.

Git

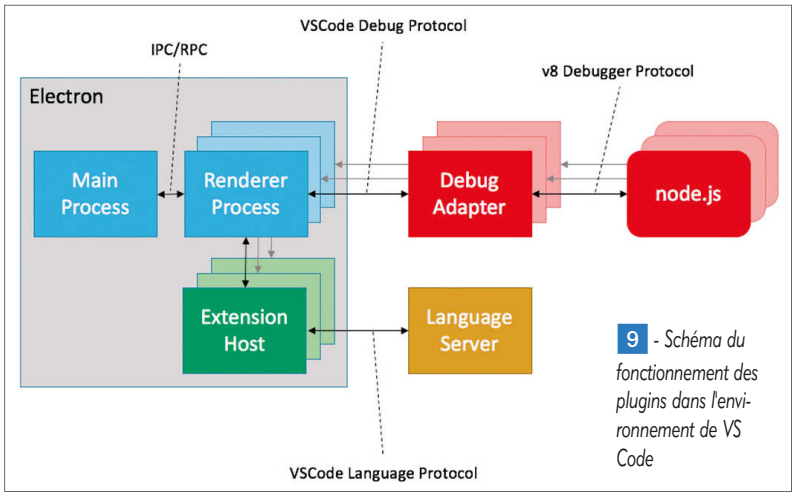
Git est devenu aujourd'hui incontournable, et Microsoft l'a réellement adopté en intégrant une extension Git automatiquement dans l'éditeur VS Code. L'extension est directement accessible via le menu de gauche.

Lorsqu'on ouvre l'éditeur VS Code sur un espace de travail sans dépôt Git, il vous propose de l'initialiser automatiquement. Il va alors lancer la commande **git init** et télécharger ainsi tous les fichiers donc il a besoin pour le dépôt. Ainsi, la plupart des commandes Git deviennent ainsi facilement accessible, que soit sur les modifications en cours (commit, push, pull …) ou sur les fichiers en eux-mêmes (voir les différences, stage …).



6 - *Commandes Git pour la gestion des fichiers*

La grande force de VS Code avec Git c'est que le contrôleur de code source est totalement intégré. Par exemple, il est facile de voir les nouveaux fichiers ou ceux en cours d'édition dans l'explorateur de fichiers, car ces derniers sont en couleurs avec une icône sur la droite pour indiquer la nature du changement. De plus, des indicateurs sur la droite de l'éditeur indiquent à quels endroits il y a des



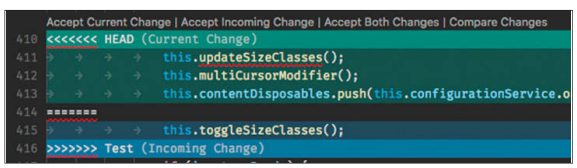
9 - *Schéma du fonctionnement des plugins dans l'environnement de VS Code*

modifications en cours, et si ces dernières sont des ajouts ou des modifications du code source. Ensuite, la branche actuelle sur laquelle se trouve le développeur est indiquée en bas à gauche, ainsi que le nombre de changements opérés en local et sur le serveur. Ainsi, d'un seul coup d'œil on peut savoir si le projet nécessite une synchronisation avec le serveur ou non.

7 - *Aperçu des rapides des modifications en cours*

La palette de commandes est également très bien fournie concernant Git et il est très facile de trouver ces dernières en commençant par écrire Git dans la palette. Par exemple, pour la gestion des branches, les commandes Git : Create Branch et Git : Checkout to permettent de créer et changer de branche facilement. D'ailleurs, lors d'un checkout, VS Code propose les branches disponibles pour aider à la saisie.

Concernant les conflits, VS Code détecte automatiquement les fichiers avec des conflits, et permet des actions inline afin de résoudre les conflits. L'éditeur met rapidement en évidence les différences, le code qui vient du serveur et le code local.



8 - *Détection des conflits par VS Code*

Il est possible d'installer d'autres extensions de contrôleur de code source dans VS Code pour gérer par exemple Subversion, TFS ou Mercurial. Ainsi, Visual Studio Code devient vraiment un outil où la productivité en devient améliorée grâce à une intégration totale et réussie des outils de contrôle de code source.

Les plugins

La gestion des plugins dans Visual Studio Code est certainement l'une des plus grandes forces de l'éditeur. Avec un écosystème de plus en plus grandissant grâce à la communauté, Visual Studio Code profite d'une richesse de plugins incroyable lui permettant de travailler avec la majorité des technologies de développement.

Le Marketplace se trouve dans le menu de gauche (*Ctrl+Shift+X*). VS Code va recommander certaines extensions selon les plus populaires et les espaces de travail ouverts. Chaque plugin est issu d'un projet GitHub : il est donc facile de contribuer ou de remonter des bugs en cas de besoin.

VS Code fournit un modèle d'intégration unique pour tous les plugins. Plusieurs notions sont importantes ici :

- L'enregistrement : comment le plugin doit s'intégrer à l'éditeur et à l'environnement de l'IDE ? ;
- L'activation : quand le plugin doit-il s'activer (ouverture de fichier, de workspace …) ? ;
- L'accession aux APIs de VS Code.

L'éditeur fournit cependant deux modules supplémentaires, selon les besoins :

- Le '**Language Server**', dans le cas où le plugin aurait besoin de fonctions avancées d'analyse de langage ;
- Le '**Debug Adapter**' si le plugin a besoin de fonctions de débogage particulières. 9

Chaque extension est exécutée dans un processus différent, ce qui permet à VS Code de rester toujours réactif. Afin de créer une extension, un générateur Yeoman est disponible à ce sujet (assurez-vous d'avoir NodeJS installé).

```
npm install -g yo generator-code
yo code
```

Il est possible d'écrire son extension en Typescript ou en Javascript. Chaque extension doit posséder un fichier **package.json** décrivant les informations principales du plugin. On y retrouve :

- **name** : le nom de l'extension ;
- **version** : la version ;
- **engines** : objet contenant au moins la clé '**vscode**'. Il permet de définir la version minimale (en mode serveur) de l'éditeur pour faire fonctionner le plugin ;
- **activationEvents** : définit une liste de commandes qui agît sur le plugin pour l'activer à des moments bien précis ;
- **main** : le point d'entrée de l'extension ;

Il existe ensuite beaucoup de propriétés pour l'aspect communication du plugin : **publisher**, **description**, **markdown**, **qna**, **keywords**, **categories** et bien d'autres.

La structure de fichier est la suivante :

- **.gitignore** : fichier à ignorer pour Git ;
- **.vscode** : paramètre de votre espace de travail VS Code ;
- **src** : dossier source de l'extension ;
- **src/extension.ts** : code de l'extension ;
- **src/test** : dossier de test. Les tests générés utilisent Mocha comme Framework de test unitaire ;
- **node_modules** : modules node installés pour l'extension ;
- **out** : dossier de génération (uniquement si l'extension est écrite en TypeScript) ;
- **package.json** : fichier manifeste de l'extension ;
- **tsconfig** : fichier de transpilation ;

Le cœur de l'extension se trouve dans le fichier **extension.ts** et expose deux fonctions importantes : **active** et **deactivate**.

```
// The module 'vscode' contains the VS Code extensibility API
// Import the module and reference it with the alias vscode in your code below
import * as vscode from 'vscode';

// this method is called when your extension is activated
// your extension is activated the very first time the command is executed
export function activate(context: vscode.ExtensionContext) {

    // Use the console to output diagnostic information (console.log) and errors (console.error)
    // This line of code will only be executed once when your extension is activated
    console.log('Congratulations, your extension "my-first-extension" is now active!');

    // The command has been defined in the package.json file
    // Now provide the implementation of the command with  registerCommand
    // The commandId parameter must match the command field in package.json
    var disposable = vscode.commands.registerCommand('extension.sayHello', () => {
        // The code you place here will be executed every time your command is executed

        // Display a message box to the user
        vscode.window.showInformationMessage('Hello World!');
    });

    context.subscriptions.push(disposable);
}
```

10 - *Code initial d'une extension*

La fonction **activate** est appelée uniquement une fois par VS Code, lors du lancement de l'extension selon les différents évènements définis dans le **package.json**. La fonction **deactivate** est implémentable afin de faire du nettoyage avant que VS Code arrête le plugin. L'extension ici ne fait qu'enregistrer une commande **extension.sayHello**. Le code dans l'expression lambda indique l'action à effectuer lorsque cette commande est appelée. C'est ici que va alors s'intégrer toute la logique de votre extension. Ensuite, il suffit de définir les lignes suivantes dans le **package.json** pour nommer et exposer la commande :

```
"contributes": {
    "commands": [{
        "command": "extension.sayHello",
        "title": "Hello World"
    }]
},
```

11 – *Corrélation entre une commande et son libellé exposé*

Ecrire sa propre extension est donc très facile, et Microsoft fournit des exemples concrets d'utilisations.

Premiers pas pour créer son langage

Il est possible via sa propre extension de créer son langage personnalisé pour VS Code. Avec quelques fichiers de configuration, il est possible d'avoir les bases comme l'auto-complétions, la coloration syntaxique ou encore l'indentation automatique. Pour ce faire, il suffit de créer un fichier **.tmlanguage** (TextMare) et de le spécifier dans son **package.json**. Le TextMate possède sa propre syntaxe pour définir les grammaires de langage et est couramment utilisé dans les éditeurs tels que Atom, Sublime Text ou VS Code.

```
"contributes": {
    "languages": [{
        "id": "mylang",
        "aliases": ["MyLang", "mylang"],
        "extensions": [".mylang",".myl"]
    }],
    "grammars": [{
        "language": "mylang",
        "scopeName": "source.mylang",
        "path": "./syntaxes/mylang.tmLanguage"
    }]
}
```

12 - *Ajout d'un nouveau langage via l'extension*

Pour vous faire une idée de la grammaire utilisée dans VS Code pour TypeScript, les fichiers sont dans un projet OpenSource : https://github.com/Microsoft/TypeScript-TmLanguage. Chaque langage est associé à un ID. Dans l'exemple ci-dessus, l'ID du langage est **mylang**, ce qui va permettre dans l'extension d'associer des commandes uniquement pour ce langage. Il est possible de rajouter ensuite d'autres fichier .json afin d'ajouter des fonctionnalités, comme les snippets. Dans le **package.json**, il suffit de rajouter les lignes suivantes :

```
"contributes": {
    "snippets": [
        {
            "language": "javascript",
            "path": "./snippets/javascript.json"
        }, ...
    ], ...
}
```

13 - *Configuration des snippets*

L'exemple ci-dessous associe au langage JavaScript les snippets contenus dans le fichier **javascript.json**. Ensuite, les lignes ci-dessous définissent le snippet que le développeur pourra utiliser par la suite dans l'éditeur.

```
"Insert ordered list": {
    "prefix": "ordered list",
    "body": [
        "1. ${first}",
        "2. ${second}",
        "3. ${third}",
        "$0"
    ],
    "description": "Insert ordered list"
}
```

14 - *Snippet défini dans javascript.json*

Les accolades définissent les endroits où le développeur va pouvoir taper du texte rapidement. L'alias **$0** définit la position finale du curseur lorsque le développeur décide de terminer sa saisie.

L'extension peut ensuite profiter d'un protocole spécial intitulé **Language Server**. Pour l'utiliser, le développeur de l'extension doit définir 2 choses :

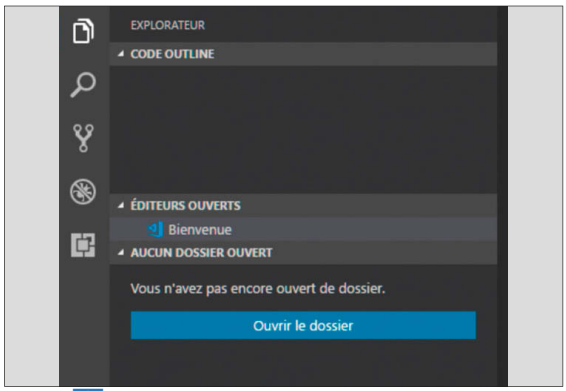
- **Le client** : c'est généralement l'extension. C'est elle qui va pousser les informations au serveur de langage selon ses besoins en analyse. Elle peut notamment communiquer avec le serveur via des objets **LanguageClient** et **LanguageClientOptions** en TypeScript ;

- **Le serveur** : c'est lui qui va traiter les demandes, valider que le texte est correct et renvoyer la réponse au client.

L'extension peut profiter d'une API écrite en TypeScript très riche se trouvant dans le paquet npm **vscode-languageserver**.

Je démarre avec VS Code

La notion importante quand on démarre avec VS Code est la notion d'espace de travail. Pour l'éditeur, un dossier est un espace de travail. Et il est donc possible d'organiser autant d'espaces de travail que l'on veut sans limitation particulière. D'ailleurs, lors de l'ouverture de VS Code, l'explorateur demande tout de suite l'ouverture d'un dossier pour commencer à travailler.



15 - *Ouverture d'un espace de travail depuis l'explorateur*

Prenons l'exemple d'une application Angular 2. Pour faire fonctionner une application Angular 2 avec VS Code, il faut :

Installer NodeJS ;

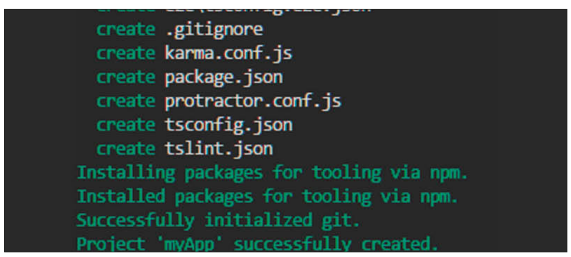
Installer Angular CLI via la commande **npm install -g @angular/cli ;**

Par la suite, nous allons expressément travailler avec le terminal intégré, ce qui permet à la fois de visualiser son code source tout en lançant les commandes adéquates au lancement de l'application.

Une fois un dossier ouvert dans VS Code, commençons par créer notre application Angular via la commande suivante :

```
> ng new myApp
```

Angular CLI va alors créer toute l'arborescence de fichiers et de dossiers pour nous.



16 - *Créartion d'une application Angular dans VS Code*

L'application étant prête, il suffit de naviguer dans l'explorateur de fichiers pour visualiser les fichiers et dossiers générés. Il est ensuite possible de créer un repository Git via l'onglet prévu au plugin. Cette action va générer un dossier **.git** avec toutes les infos dont le gestionnaire de code source a besoin.

Ensuite, toujours dans le terminal intégré, il suffit d'utiliser la commande suivante pour lancer l'application :

```
> ng serve
```

Un navigateur va automatiquement se lancer et l'application va se charger. Toutes les modifications effectuées dans l'éditeur sur un composant Angular vont automatiquement être rechargées dans le navigateur grâce à Angular CLI.

17 - *Compilation Angular dans Visual Studio Code*

Ainsi, avec Visual Studio Code nous avons réussi à créer une application Angular à partir de zéro et à la lancer depuis l'éditeur. Ceci permet d'avoir un environnement intégré et complet permettant à la fois d'interagir avec le code et de lancer les commandes nécessaires pour exécuter, compiler, tester et débugger.

Monaco, l'éditeur VS Code dans votre application

VS Code utilise un éditeur de texte OpenSource intitulé Monaco. Il a été spécialement conçu pour VS Code et est disponible sur GitHub pour que n'importe quel développeur puisse intégrer cet éditeur dans son application Web : https://github.com/Microsoft/monaco-editor. Intégrable dans une application JavaScript comme TypeScript, cet éditeur embarque de base plusieurs fonctionnalités très intéressantes comme :

- Validation de langages comme TypeScript, JavaScript, CSS, LESS, HTML, JSON … ;
- Coloration syntaxique basique ;
- Comparaison en live de fichiers ;
- Gestion du Rechercher / Remplacer ;
- Gestion des commandes simples ;
- Numéros de lignes ;
- Le *Go To* ;
- Le *Rename* ;
- Les rapports d'erreur.

Ce package permet déjà de répondre à bon nombre de scénarios simples d'un éditeur de code classique. Avec ceci, le développeur est également capable d'intégrer ses propres commandes et snippets de code dans l'éditeur, selon ses cas d'usages. Il peut par exemple coder son propre **Ctrl+S** pour la sauvegarde de ses fichiers. Grâce au moteur Monarch sous-jacent, le développeur est même capable de créer ses propres syntaxes et colorisations selon ses besoins.

Conclusion

OpenSource jusqu'à rendre disponible son éditeur de texte en libre accès, Visual Studio Code incarne le renouveau de Microsoft au travers des outils à destination des développeurs. Il intègre toutes les fonctionnalités d'un éditeur moderne, tout en reposant sur une architecture robuste afin de garantir la stabilité et la performance de l'outil. Le système de plugins, point fort de l'éditeur, permet à une communauté toujours plus grandissante d'apporter sa pierre à l'édifice en proposant de nouvelles fonctionnalités chaque jour. •

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...

Choisir votre base de données

Une base de données est une brique indispensable dans une application, et les possibilités proposées sont très larges car vous pouvez trouver de nombreux formats comme le SQL, NoSQL, développement web, scalabilité, architecture, hybride, base objet. Ce qui ne simplifie pas votre choix ! 1

INTRODUCTION

Avec plus de 300 bases reconnues et identifiées par le site DB-Engines (https://db-engines.com), il est difficile de faire un choix entre les bases de données historiques et les bases de données dites « modernes ». Nous allons essayer d'éclaircir le paysage.

Les bases de données se décomposent en plusieurs familles : tout d'abord, celles développées par des sociétés (ex. : Oracle, IBM, Microsoft) et celles venant des fondations ou des communautés (MySQL, MariaDB). Mais le choix final ne peut pas se limiter à ces 2 critères car il faut regarder le modèle de données, le support, la formation, le coût et surtout l'usage que l'on veut en faire. 2

LE MODÈLE DE DONNÉES

SGBDR

Le SGBDR (Système de Gestion de Bases de Données Relationnelles) est un logiciel système destiné à stocker et à partager des informations dans une base de données. Cette technologie repose sur la théorie mathématique des ensembles et peut arriver au plus petit composant manipulable de type atomique.

Par exemple Monsieur Martin Dupont, sont des critères très rarement stockés dans un même champ. Si vous souhaitez isoler les personnes ayant comme civilité « Monsieur » et trier par ordre alphabétique les noms suivis de leurs prénoms. Vous devez stocker les données du nom dans trois colonnes différentes d'une table.

Ainsi, les bases de données relationnelles offrent à la plupart des applications, qu'elles soient opérationnelles ou analytiques, des performances suffisantes en matière de stockage, d'accès, de protection et de traitement. Elles garantissent la qualité, la pérennité et la confidentialité des informations tout en cachant la complexité des opérations.

Les fonctionnalités basiques permettent d'enregistrer les données, de faire des recherches, de les modifier et de retourner un résultat, mais le coût peut énormément varier d'une base de données à une autre car vous avez le choix entre une base de données propriétaire ou une base de données Open Source.

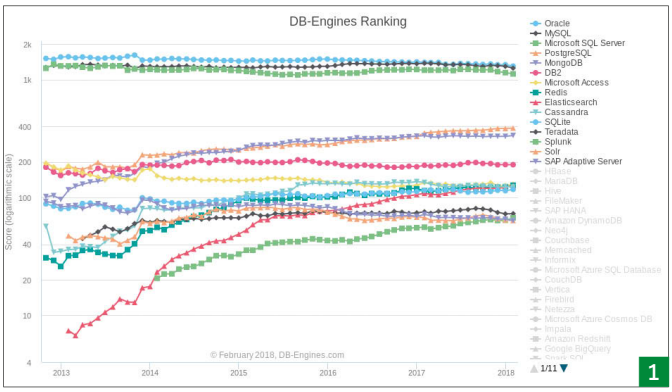
L'approche est différente, mais reste dans la même logique. La différence correspond aux coûts qui seront inférieurs pour les solutions Open Source (NDLR : la licence est le coût le plus visible). Toutefois, il est important de bien étudier votre usage car certaines proposent le minimum et vous risquez d'arriver rapidement à une limitation fonctionnelle bloquante.

Bien entendu, le SGBDR est un élément central de la plupart des systèmes et applications informatiques. Il va continuer à dominer le paysage des bases de données car il est reconnu comme fiable, même si celui-ci évolue.

NoSQL

Les bases de données dites NoSQL (Bases de données NON relationnelles ou Not only SQL) n'utilisent pas le modèle relationnel et n'utilisent pas le classique langage SQL.

Elles sont destinées principalement à gérer un volume important de données qui nécessite une grande montée en charge avec une tolérance aux pannes. Le schéma de données est très simple, voire inexistant, c'est pourquoi il est très utilisé dans des environnements Cloud ou Big Data. Mais nous pouvons le retrouver dans l'analyse de



1

Rank Feb 2018	Rank Jan 2018	Rank Feb 2017	DBMS	Database Model
1.	1.	1.	Oracle	Relational DBMS
2.	2.	2.	MySQL	Relational DBMS
3.	3.	3.	Microsoft SQL Server	Relational DBMS
4.	4.	4.	PostgreSQL	Relational DBMS
5.	5.	5.	MongoDB	Document store
6.	6.	6.	DB2	Relational DBMS
7.	7.	↑ 8.	Microsoft Access	Relational DBMS
8.	↑ 9.	↑ 10.	Redis	Key-value store
9.	↑ 10.	↑ 11.	Elasticsearch	Search engine
10.	↓ 8.	↓ 7.	Cassandra	Wide column store
11.	11.	↓ 9.	SQLite	Relational DBMS
12.	12.	12.	Teradata	Relational DBMS
13.	↑ 15.	↑ 16.	Splunk	Search engine
14.	14.	14.	Solr	Search engine
15.	↓ 13.	↓ 13.	SAP Adaptive Server	Relational DBMS
16.	16.	↓ 15.	HBase	Wide column store
17.	17.	↑ 20.	MariaDB	Relational DBMS
18.	18.	↑ 19.	Hive	Relational DBMS
19.	19.	↓ 17.	FileMaker	Relational DBMS
20.	20.	↓ 18.	SAP HANA	Relational DBMS

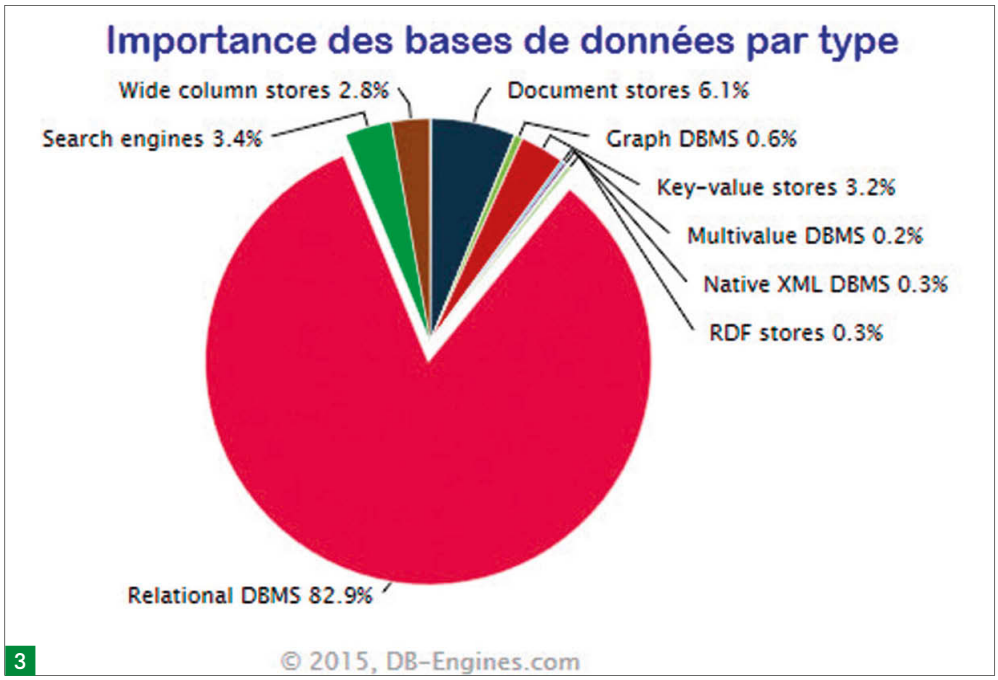
2

performances venant des Logs. Ces bases sont utilisées pour les données non structurées.

Exemple : l'utilisation des fichiers à plat comme le NoSQL permet de stocker des fichiers de type image ou audio directement dans la base de données sans altérer la structure de celle-ci. Il devient plus simple de placer des données dans la base de données grâce à la souplesse des données qui ne sont pas figées.

Ce format regroupe des systèmes très divers venant de bases de données différentes sous différents noms de moteur de stockage comme ColumnStore, Wide Column Stores, Document Stores, Graph DBMS, RDF Stores, DBMS, Search Engines...

Par rapport au format SGBDR, il a une approche horizontale pour enregistrer les données de manière non structurée dans



3

une base de données, c'est à dire sans relations, sauf si celles-ci ne sont pas déjà établies avec les autres données du même système. Cette relation ne se crée que lorsque vous transformez les données dans la base de données.

In-memory

Les bases de données au format « in-memory » ont une capacité de stocker en mémoire des structures entières de bases de données. L'avantage de cette technique est de soulager les opérations sur disques pouvant être plus longues à traiter avec des gros volumes de données. Le résultat obtenu est une réduction des temps de réponse entre l'application et la base de données. Ainsi, vous pouvez trouver des bases de données dédiées à ce format comme VoltDB, Aerospike, SAP Hana, MariaDB. Elles peuvent interagir avec l'ensemble des langages. Cette technologie fait partie de la nouvelle génération de bases de données, pouvant aussi classer en « NewSQL », permettant de structurer plus facilement les données. Elles sont très appréciées pour les architectures innovantes qui sont utilisées pour traiter des volumétries importantes de données comme le BigData avec la possibilité de les structurer.

Ce système est utilisé principalement pour améliorer les performances des requêtes et des applications qui accèdent aux données. Il peut également bénéficier d'un jeu d'instructions réduit compte tenu de la réduction du nombre d'activités ayant besoin d'accéder aux données (à la différence d'un accès à partir d'un disque).

Toutefois, les autres bases de données sont en train d'implémenter cette technologie dans leurs environnements ; il faudra vous rapprocher de leurs géniteurs pour connaître à partir de quelle version elles peuvent être disponibles.

Leur utilisation est de plus en plus intéressante, car la puissance des machines augmente et permet de garder les données en mémoire, même après une panne. Grâce à une fiabilité et une persistance des données améliorée, cette architecture répond désormais aux besoins en traitement transactionnel et analytique de la plupart des entreprises. 3

LA RÉALITÉ

Dans le milieu professionnel, le choix peut sembler difficile car vous effectuerez au préalable (du moins si le projet est bien mené) une étude pour définir le meilleur scénario pour les données. Ceci en estimant la volumétrie de données et son nombre d'enregistrements en lecture/écriture par heure.

Toutefois, la sélection d'une base de données par rapport à une autre varie suivant les fonctionnalités attendues et dans les cas d'usage qui feront pencher la balance. Par exemple si votre base de données est à destination du web, du mobile, de l'embarqué, du cloud…

Bien entendu, les projets d'aujourd'hui ont souvent une base de données relationnelle pour les données à fort besoin d'atomicité et d'intégrité, et une (ou des) base NoSQL pour les données à forte sollicitation en lecture en de très gros volumes. Ces bases sont ensuite interfacées/synchronisées pour le maintien des données communes.

Projet avec du NoSQL

Les bases NoSQL sont nombreuses même si on en cite généralement que 2 ou 3 comme MongoDB, Cassandra ou Elastic Search. La première (populaire) a été pensée pour supporter de gros volumes de données. Elle implémente une logique de scalabilité horizontale avec du sharding. Les dernières permettent d'effectuer de la recherche avancée (géospatiale, multicritères). Le format de stockage sur le disque est un format BSON (Binary + JSON) permettant un gain de performance et d'espace disque. Toutefois, il existe certaines solutions complémentaires comme l'interface de l'API Rest qui n'est pas supportée nativement.

Un autre point à ne pas négliger concerne le paramètre d'indexation qui n'est pas possible actuellement, limitant l'utilisation de certaines requêtes comme la recherche entre documents de manière native. Le développeur devra effectuer 2 requêtes, par exemple une première pour retrouver le document et récupérer l'object id du second, et une deuxième pour retrouver le document ayant l'object id.

Cassandra est destinée à une utilisation intensive du NoSQL, comme de nombreux sites à fort trafic. Ainsi, il permet de réaliser une forte scalabilité, et de garantir une haute disponibilité. De plus, il est possible d'ajouter un nœud Cassandra dans un cluster pour améliorer sa puissance. Il n'y aura pas de perte de performance par rapport aux autres. Toutefois, un schéma doit être défini lors de la conception, car le système est orienté en colonnes avec la présence de clefs d'index.

Elastic Search est un moteur de recherche orienté documentaire, et il s'appuie sur Lucene, permettant de bénéficier de très hautes performances au niveau de la recherche complexe sur les gros volumes de données.

Cette base de données a été développée avec l'optique de faire un moteur « no SPOF » (no Single Point Of Failure). Le

principe, c'est que dans un cluster de plusieurs bases, même si un nœud vient à s'éteindre comme un crash serveur, la donnée sera toujours disponible et le service continuera de fonctionner.
Toutefois, cette base de données n'est pas prévue d'être la base de données principale, car elle se positionne comme moteur de recherche et non comme une base de données. De plus elle ne gère pas la relation entre les documents sans faire 2 requêtes.
Enfin, il existe de nombreuses bases de données NoSQL répondant à une problématique précise qui ne convient généralement pas à l'ensemble d'un projet.

Fork ou original

Un fork est un logiciel créé à partir du code source d'un logiciel existant lorsque les droits accordés par la licence le permettent. Le cas le plus connu est MySQL. Depuis le rachat par Sun, puis par Oracle, nous avons vu apparaître de nombreux projets, dont les plus connus sont : MariaDB et Percona.
MariaDB est développée par la fondation MariaDB. Elle propose de nombreuses améliorations et d'évolutions au niveau des fonctionnalités pour les développeurs et les moteurs de stockage (storage engine). Ainsi, depuis plusieurs années, de nombreuses distributions Linux utilisent cette base de données. Avec ses nombreuses évolutions, elle a répondu aux attentes des professionnels et se positionne avec une solution orientée NewSQL. Ceci en regroupant le meilleur des mondes SQL et NoSQL, mais aussi en ayant répondu aux problématiques modernes, de clustering et de réactivité.

La réactivité

La réactivité et les temps de réaction sont importants. Les processus de réactivité au niveau de la sécurité sont très courts quand la base de données s'appuie sur la communauté par rapport à une base de données historique avec des processus plus longs. En contrepartie, elle garantisse un niveau de support et de processus qui rassurent certaines entreprises.

LES PLATEFORMES

Une base de données est avant tout un outil de collecte de données. Quelle que soit l'utilisation que vous effectuez, elle est liée à votre projet et une utilisation quotidienne ; vous devez prévoir un espace de

4

stockage. Pour cela, un des premiers critères s'oriente sur l'ancienneté de celle-ci, si un support ou une assistance existe. De plus il est intéressant de regarder si celle-ci bénéficie d'évolutions et d'améliorations régulières, avec la présence d'une communauté, de contributeurs ou de sociétés qui l'utilisent.
Il est important de faire attention à votre choix, car une base de données peut apparaître et disparaître quelques années pour de nombreuses raisons.
L'architecture de la base de données déterminera la ou les plateformes adaptées. De manière générale, l'enregistrement de données non structurées que nous appelons « base de données horizontale » s'orientera vers le NoSQL.
L'approche verticale aura une approche structurée de la donnée, c'est-à-dire qu'elle sera transformée avant d'être regroupée, afin que la bonne relation soit immédiatement établie entre les données au sein de la base de données. Chacune d'elle présente ses avantages et inconvénients car elle influence vos systèmes.
L'hébergement des bases est multiple : serveur mutualisé ou dédié, dans le cloud (IaaS ou sous forme de service).
Avec les solutions cloud, il est possible de couvrir la majorité des demandes, permettant de retrouver des interfaces modulaires, que vous pouvez choisir. Toutefois, l'ensemble des bases de données ne sont pas disponibles nativement chez tous les hébergeurs, mais les bases de données standard le sont. Ainsi, vous pouvez définir votre façon de faire du cloud : privé, public, hybride Toutefois, une nouvelle façon d'héberger les données est devenue assez populaire, appelée DbaaS (Database as a Service, une solution de plateforme externe dans le Cloud). 4

Conclusion

Chaque base de données possède des avantages et des inconvénients, avec de nombreux critères à prendre en compte. Tout d'abord, quels que soient vos choix, vous utiliserez de la mémoire ou du cache pour traiter, gérer et manipuler vos données. Ensuite, le stockage est aujourd'hui un faux problème, car l'espace varie soit sur le coût de la mémoire, soit sur le coût du matériel. Toutefois, la performance peut s'en ressentir lors de fort trafic car la robustesse sera mise à l'épreuve.
Bien entendu, le secteur des bases de données d'aujourd'hui et de demain, restera très volatil car même si l'écosystème SGBDR domine, le marché est beaucoup plus complexe qu'il n'y paraît. •

Thierry Leriche-Dessirier
Architeche
http://www.icauda.com

Retour d'expérience dans le choix d'une base de données NoSQL pour un projet haute dispo/perf

À l'occasion de la refonte du système de réservation d'un grand groupe hôtelier, le choix de la base de données a été au cœur des préoccupations. Les enjeux et les rebondissements, pas seulement techniques, ont été nombreux. Voici le retour d'expérience de cette aventure.

Début 2015, notre hiérarchie nous propose un nouveau défi : faire face à l'augmentation de l'utilisation des sites et outils du groupe. En effet, la demande mondiale croît de façon constante et l'ancien SI vieillissant ne sera bientôt plus en mesure d'absorber la charge. Le projet est donc stratégique.

Situation de départ

L'ancien SI s'appuyait sur un cluster Sysbase de 75 gros serveurs répliqués. La logique métier était massivement implémentée à l'aide de procédures stockées. La refonte précédente, développée en Java 5, avait conservé ce cluster et s'était soldée par un échec. Les serveurs étaient à bout de souffle et les temps de réponse étaient déjà dégradés au-delà du raisonnable. La situation devenait critique…

Ajouter de nouveaux serveurs n'offrait que peu de puissance supplémentaire, pour un coût en énergie et en argent prohibitif. En outre, il devenait de plus en plus difficile de trouver des développeurs en mesure de maintenir le système. Cette architecture ne convenait manifestement plus.

Objectifs

Avec une présence mondiale, les systèmes sont en activité 24/7. Le gros de l'activité se concentre toutefois en France avec un plateau (pic) d'une dizaine d'heures. Le système doit être disponible à 99,9%, soit une indisponibilité max de 8,76 heures par an, ou 10 minutes par semaine.

Les performances sont de toutes les discussions. Le web service principal doit répondre en 90ms au 95e centile, avec un objectif caché de 40ms(1), et en 150ms au 99e. À l'échelle de la charge attendue, le centile restant n'est déjà plus négligeable. D'ailleurs, à ce niveau de performance, chaque milliseconde gaspillée pèse lourd sur la balance. La charge nominale prévue pour ce même web service (dix heures par jour) sera de 1500 req/s dès 2016, puis 2400 en 2017 et 3200 en 2018. Les estimations sur l'augmentation du trafic s'étant toujours vérifiées durant la dernière décennie, ces volumes sont donc réputés fiables. Le système devra donc être élastique.

> « LES MILLISECONDES GASPILLÉES COÛTENT CHER, IL FAUDRA INTERVENIR À TOUS LES NIVEAUX… »

Enfin, en raison du fonctionnement du groupe et de ses entités, on attend environ 300 millions de mises à jour diverses (prix, disponibilités, produits, etc.) par jour, soit plus de 3000 par seconde en continu. Des techniques de conflation, similaires à celles utilisées dans le secteur boursier, permettent heureusement de réduire ce chiffre.

Candidates

Sans surprise, les solutions SQL classiques ont été écartées. Certaines d'entre elles auraient pourtant été intéressantes, mais l'expérience avec l'existant refroidissait les équipes. Cela fait aussi partie des critères de choix. La mode étant alors au NoSQL, c'est donc sur cette voie que nous avons concentré nos réflexions.

Plusieurs solutions nous ont été proposées (imposées) par notre hiérarchie, et nous avons complété la liste avec les stars du moment, pour lesquelles nous avions de bons retours. Nous étions ouverts aux différentes typologies, sans a priori, pour répondre au mieux à nos contraintes.

Cette première liste contenait notamment (liste non exhaustive) MySQL Cluster(2), MongoDB (document), Hazelcast (Grid), Cassandra (colonne), CouchBase (document) ou encore Neo4J (graphe). La proposition d'Hazelcast peut surprendre, mais elle est pourtant pertinente.

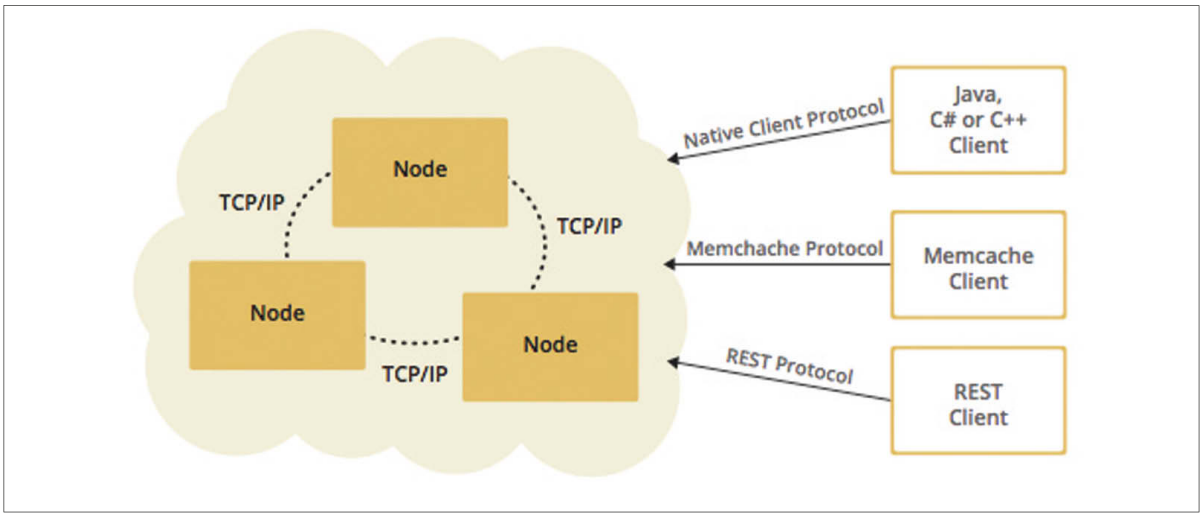
Critères de choix

La base de données doit être performante, en particulier pour la lecture que nous avons privilégiée. C'est le critère principal. Le système doit pouvoir l'attaquer de manière asynchrone, ce qui élimine d'office JDBC. Les transferts réseau sont également concernés. La quantité d'informations remontée étant conséquente, un mécanisme de compression doit être intégré.

Le cluster doit être résiliant aux pannes. Cela ne signifie pas que les instances individuelles ne puissent pas subir d'avarie, mais que, le cas échéant, le cluster sera en mesure de les compenser, sans réduction de service.

Un mécanisme de redondance de l'information est bien entendu indispensable. Le système étant distribué sur plusieurs conti-

(1) Cet objectif ne sera finalement pas atteint en charge max. (2) A ne pas confondre avec MySQL.



Hazelcast Server Member

nents, le cluster doit permettre une synchronisation multi-sites, si possible en temps réel.

Le cluster doit être élastique : on doit pouvoir ajouter des instances pour faire face à l'augmentation de la charge. La prise en compte des nouvelles instances ainsi que la redistribution des données doivent être transparentes et rapides.

Les matériels utilisés, et en particulier les serveurs, doivent être standards. Il n'est pas question d'investir dans des machines à plusieurs centaines de milliers d'euros. Et puisqu'on parle du prix, celui des licences doit être raisonnable (et négociable). Nous choisissons systématiquement des licences et supports en version entreprise.

Enfin, et ce n'est pas le moindre des critères, la base doit être connue des développeurs et des DBA(3), quitte à assurer une (petite) formation complémentaire.

Solution retenue

Plusieurs bases candidates ont rapidement été écartées conformément aux critères exposés ci-dessus. L'une d'elles, imposée et dont on taira le nom, s'est d'ailleurs révélée très décevante, à mille lieues des promesses commerciales. Les bases graphes ne correspondaient finalement pas à notre domaine, sans le torturer. Le critère d'élasticité a, quant à lui, éliminé une bonne partie des candidates.

Cette première sélection s'est conclue avec différents dossiers d'architecture et présentations auprès des autres services impliqués et de notre hiérarchie. Il ne restait plus alors en lice que CouchBase, Cassandra et Hazelcast, que nous avons testé et analysé plus en profondeur, et qu'on pourrait conseiller à de nombreuses équipes.

Nous avons travaillé main dans la main avec les développeurs et architectes de ces solutions. Après six mois d'étude, nous avons choisi Cassandra Entreprise, avec le support de la société Datastax, pour la base de données. Nous avons également intégré Hazelcast Entreprise pour ses fonctionnalités de cache distribué et de Grid Computing.

> « IL A FALLU DÉFENDRE NOS CHOIX… »

Mise en œuvre

La suite du projet s'est étalée sur huit mois. Nous avons rapidement installé Cassandra sur nos environnements (dev, intégration, préprod, pro) et sites avec le support des ingénieurs de Datastax.

Le coefficient de réplication a été réglé sur 3 : les données sont écrites sur un nœud puis répliquées de manière asynchrone sur deux autres nœuds. Le dimensionnement initial du cluster, avec ses 16 instances, a été provisionné pour les deux premières années. Les machines physiques ont été reliées par fibre optique à des équipements dédiés.

Cassandra confirme l'enregistrement des données après les avoir sérialisées sur le support de stockage et non seulement en mémoire, contrairement à d'autres bases. Ce point peut s'avérer important à forte charge en cas de défaillance. Pour des raisons de performance évidentes, Datastax recommande de préférer le SSD aux disques rotatifs. Des algorithmes distincts sont employés selon le support.

Les processus sont distribués sur le cluster Hazelcast d'où ils attaquent le cluster Cassandra. Après avoir testé plusieurs ORM, dont Achille et Hibernate, nous avons décidé de travailler directement en CQL (Cassandra Query Language) pour des raisons de performance.

Cassandra propose plusieurs niveaux de consistance, impactant le nombre de nœuds devant être sollicités pour lire une donnée, dont All, Quorum, One, Any, etc. Dans notre cas particulier, les processus de lecture et d'écriture étant fortement décorrélés, nous avons choisi Any : le nœud le plus rapide fait foi.

Nous avons consacré beaucoup de temps pour concevoir le modèle et il a fallu s'y reprendre à plusieurs fois. Dès le départ, nous nous sommes appuyés sur la fonctionnalité « time series » car nos données étaient conditionnées par la date. N'oublions pas que NoSQL signifie qu'on ne fonctionne plus en SQL classique, mais pas qu'on s'interdit complètement d'en faire. Pour certaines parties, nous avons donc normalisé les données.

Conclusion

Six mois pour choisir une base de données, cela peut sembler long, mais c'est un investissement sur le long terme. Nous avons posé les fondations de la nouvelle stack technique du groupe, et pas seulement pour ce qui concerne la base. Nous avons choisi et défendu Cassandra car cette solution de type Colonne était la plus adaptée à nos besoins spécifiques, et nous ne l'avons jamais regretté. •

(3) Administrateur de Base de Données.

Questions – réponses avec Stéphane Mahmoudi de MarkLogic

Aujourd'hui, la gestion de données non structurée est hétérogène que ce soit On-premise ou Cloud, quelle est votre approche pour casser les dépendances et les silos ?

Effectivement, par défaut, que la donnée soit sur le Cloud ou On-premise importe peu. C'est un choix d'infrastructure qui a plus à voir avec une question de coût de stockage. Par défaut seulement car le Cloud impose d'avoir des données protégées, ce qui peut être plus couteux que dans un environnement On-premise déjà sécurisé.

Les silos n'existent pas seulement avec les données non structurées. Pour des raisons souvent très justifiées, les entreprises ont des informations sur leurs clients, leurs produits, etc. qui peuvent être plus ou moins riches dans les différents silos qui constituent le système d'information.

Dans la base CRM, le client est défini par certains attributs, dans la base de commande, le client peut être défini avec des attributs différents.

Si les silos existent, c'est qu'ils ont des applications, donc des utilisateurs ou des process, qui en ont besoin. Se pose alors le problème de consolider les silos (combine de temps), tout en continuant d'avoir des applications opérationnelles.

Le solution pour « casser » les silos tout en les laissant opérationnels passe donc par :

- Une possibilité de décrire les données (métadonnées)
- Une possibilité de lier les données entre elles : par exemple, le client est lié à un contrat, à un bon de commande lui-même lié à un ou des produits, etc. Il peut avoir un compte sur des réseaux sociaux. Bref la possibilité de modéliser le monde réel caractérisant son client, ses informations et ses actions par rapport à l'entreprise.
- Une sécurisation de la donnée :

En effet, le fait de mettre des données ensemble va permettre de les consommer par différents types de consommateurs. Il faut avoir la capacité de dire quelle donnée peut être consommée par qui (ou par quoi, si c'est un process qui l'utilise).

En Cloud, il existe plusieurs solutions pour stocker et traiter les données non ou semi structurées (datalake, base objet, etc.), quelles sont vos préconisations ?

La première question à se poser est de savoir ce que l'on veut faire des données (qu'elles soient structurées, semi structurées ou non structurées).

- Est ce que l'on veut les données à des fins de reporting ?
- Est ce que l'on veut les données pour alimenter d'autres applications ?

La seconde question à se poser est comment automatiser la collecte des données.

- Est ce que c'est un système de push que l'on va mettre en œuvre,
- Est ce que c'est un système de pull (avec des engines sur la sources) que l'on va mettre en œuvre.

En réalité, les Data Lake lorsqu'ils ne sont pas survendus correspondent bien à la partie collecte des données, beaucoup moins à ce que l'on va faire des données ensuite.

Le Data Lake s'impose lorsque la collecte est complexe due à l'existence de nombreuses sources de données et qu'elle doit être automatisée.

Ensuite l'utilisation, en complément, d'une base de données capable de récupérer les différentes sources, formats et de les lier ensemble paraît indispensable si on veut créer des applications de façon simple et rapide. La création d'applications sur les Data Lake est complexe et peu productive. Le data scientist ou le développeur passe son temps à transformer la donnée pour pouvoir l'utiliser fonctionnellement alors que sa valeur ajoutée est dans le fonctionnel et non dans la transformation.

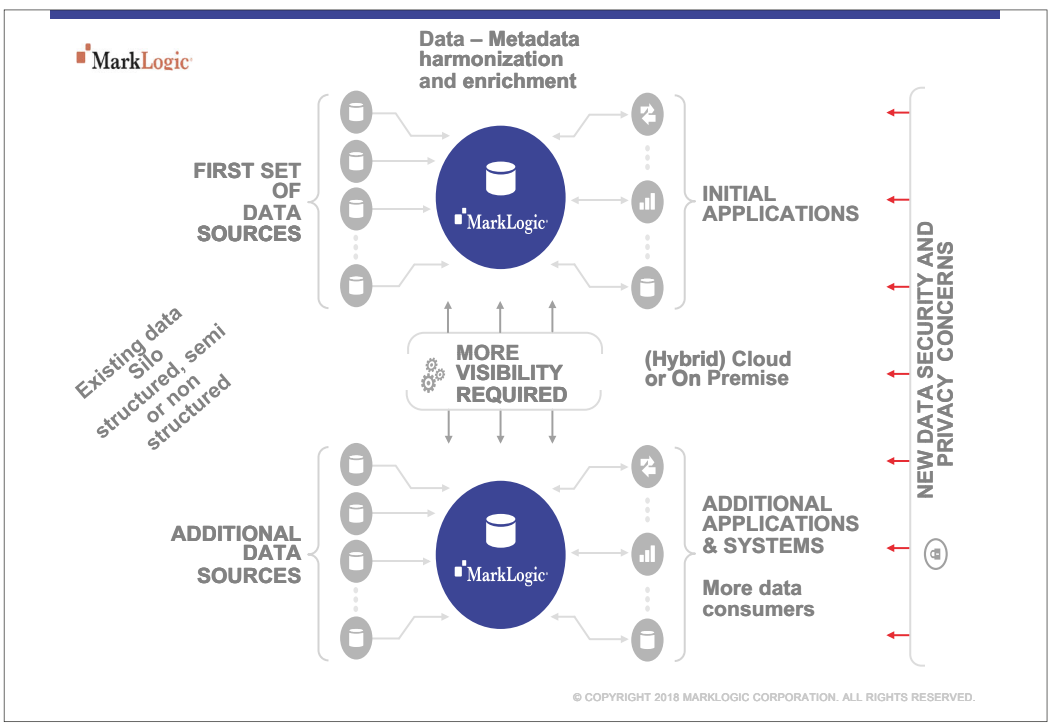
Le choix d'une base NoSQL n'est jamais simple. De par votre expérience, quelles sont les bonnes pratiques ou les conseils à donner ?

Effectivement, le choix n'est pas simple car il y en a beaucoup sur le marché.

- Les bases orientées colonnes ou clé valeur ;
- Les bases orientées document ;
- Les bases graphes ;
- Les bases multi-modèles ;

De manière générale, il ne faut pas oublier les critères suivants :

- Les données doivent être sécurisées y compris lorsqu'elles sont manipulées par un administrateur,
- Selon l'utilisation, elles doivent permettre de faire des commit et des rollback fiables et robustes,
- Elles doivent avoir des mécanismes de fail-over et de Disaster Recovery pour être en phase avec les standards des départements de production des entreprises. •



François Tonic

Il n'existe pas de choix défini

"Choisir son SGBD". Le titre du dossier est ambitieux car comme vous le savez, choisir un outil technique n'est jamais simple : il est peut-être imposé par un client, une entreprise ou l'existant. Si vous devez reprendre un site web ou app mobile, la base de données est déjà choisie et en production. Sauf à migrer, vous n'aurez pas le choix.

Nous vous proposons une liste de critères à garder en tête dans vos réflexions. Ce n'est qu'une exquisse de matrice décisionnelle, à vous de la compléter et de l'adapter à vos besoins. N'oubliez jamais qu'une base choisie pour un projet A n'est pas forcément adapté à un projet B. Et attention à ne pas générer une adhérence trop forte entre votre app et la base proprement dite. Plus les dépendances entre les couches seront fortes, plus vous aurez de difficultés à mettre à jour et à migrer !

Quelques réflexes peuvent aussi vous aider :

1 identifier et évaluer précisément vos besoins (= besoins réels) tout en se projetant vers l'avenir et les évolutions possibles. Il ne faut pas que votre choix d'aujourd'hui empêche toute évolution future ;
2 définir le ou les modèles de données ;
3 quelle est l'architecture prévue, ou l'infrastructure ?
4 quelles sont les contraintes ?
5 pérennité et évolution de la base qui sera choisie.

On pourra aussi se poser les questions suivantes :

- les structures de données sont-elles fixées ou évolutives ?
- ai-je besoin d'une grande capacité de stockage (et de traitements) ?
- quelle est la volatilité de mes données ?
- Nécessité de l'atomicité (ACID) ou non ?
- Quelles critères de performances (attention : de manière objective et réelle)?

Un critère non technique sera aussi l'attitude d'un éditeur de base ou des prestataires. La relation doit être saine et non hostile. Une agressivité des commerciaux n'est jamais une bonne chose et l'éditeur / prestataire peut alors subir un *persona non grata*.

Le critère	Type de base	Commentaires
Le coût	Tout type	Ce critère est un des plus visibles mais pas forcément le plus pertinent. Car une base open source sera généralement sans licence sauf en version entreprise. Mais attention, vous devez considérer le coût direct (licence, type de licensing, les fonctions optionnelles) et les coûts indirects que l'on oublie trop rapidement (support, modules optionnels, mécanismes avancés bridés, etc.).
Support	Tout type	Ne négligez pas le support technique et utilisateur. C'est un vrai différenciant. Qu'est-ce qui prévu en standard et en option? Quel niveau de support ? Français ou Anglais ? Est-ce l'éditeur ou des prestataires ?
Quels types de données dois-je gérer et intégrer ?	Typiquement SQL ou NoSQL	Aujourd'hui on parle de données structurées, semi-structurées, non structurées. Une base SQL n'est pas faite, initialement, pour gérer du non-structuré. Les données sont-elles homogènes ou provenant de sources peu maîtrisées, ou de natures différentes ? Est-ce des données chaudes ou froides (ex. : en architecture lambda ou équivalent, nous allons avoir des données froides de type batch et des données chaudes de type stream)? Vous n'allez pas prendre une base NoSQL si vous faites du structuré cela n'aurait aucun sens. Ni prendre un stockage objet pour du structuré pur et dur.
Open Source ou commercial ?	Tout type	Ce débat n'est pas récent. C'est à vous d'évaluer et de tester les solutions. La base open source (MySQL, MariaDB par exemple) suffira à de nombreux projets. Certaines fonctions très avancées (déduplication, cryptage de bout en bout, architectures spécifiques) peuvent être absentes ou limitées dans les solutions ouvertes mais aussi propriétaires.
In-Memory est-il indispensable ?	Tout type	La base en mémoire est aujourd'hui répandue. Mais est-elle pour autant indispensable à vos projets ? Quels sont vos critères de performances, d'accessibilité aux données ou encore de temps de traitement exigés ? Pour un site web ou même une app mobile standard, le In-Memory est-il réellement indispensable ? Idem pour des traitements batchs sur vos données.
Sécurité	Tout type	La sécurité doit être votre souci n°1. Etudier les mécanismes de sécurité disponibles dans la base choisie. N'hésitez jamais à comparer. Dois-je crypter de bout en bout ? Les données sont-elles chiffrées en stockage ? Les connexions sont-elles sécurisées ? Est-ce que des contrôles d'intégrité sont possibles durant les transferts ? Quels sont les failles connues dans le SGBD utilisé ? Etc.

Le critère	Type de base	Commentaires
De quel type de base ai-je besoin ?	Tout type	Question bête mais tellement importante. Au-delà des données que je dois récupérer et stocker, il faut se poser la question de l'analyse et des traitements que je vais y faire. Car si on parle de base relationnelle, NoSQL, objet, graph, SQL, etc. l'usage derrière va être important. Me faut-il une base clé/valeur (le clé est connue) ? Me faut-il une base orientée colonnes (la donnée sera stockée en colonnes et non en rangée) ? Quid de la base dite graphe (base utilisant la théorie des graphes) ? Quelles volumètries de données sont attendues ?
Base locale (serveur typiquement), hébergée, en mode cloud (IaaS ou service à la demande) ?	Tout type	Les nouvelles architectures bousculent nos habitudes : cloud, bases à la demande, Big Data, IoT, conteneurs…
Structurée ou non structurée ?	Tout type ou presque	C'est une des questions essentielles à se poser. Selon le type de données, vous allez vous tourner vers un base classique ou non. Question bête mais toujours utile à poser.
Réplication, déduplication, montée en charge, clustering	Tout type ou presque	Parmi les fonctionalités avancées nous trouvons la réplication, la déduplication des données, la capacité de monter en charge, le déploiement et la gestion du clustering de données. Ces fonctions nécessitent parfois des éditions spécifiques (entreprise, datacenter par exemple), ce qui peut alourdir la facture.
Migrer d'une base locale à une base cloud	Tout type	Aujourd'hui, il existe de très nombreux services de bases de données à la demande ou en mode IaaS. Les grands éditeurs de base proposent des éditions en ligne (Oracle, IBM, Microsoft pour ne citer qu'eux). Les fournisseurs cloud proposent soit des services de données à eux ou des bases tiers.
Base pour app mobile	Typiquement SGBDR	L'explosion des apps mobiles a bouleversé le monde des bases. Là aussi vous aurez du choix : base sur le device, service à la demande (type Backend as a Service) ou connexion à des serveurs. Si vos apps exigent peu de ressources de données, optez pour une base installée sur le matériel. Pour des traitements plus lourds ou des fonctions plus avancées, un service BaaS sera souvent conseillé, notamment pour le stockage des données, par exemple si vous faites des fonctionnalités cognitives.
Support plateforme	Tout type	Votre base a-t-elle des contraintes de systèmes ? Doit-elle fonctionner sur Linux ou Windows ou les deux ? Vous avez un historique mainframe et vous devez le conserver, quid de la base ? Si vous passez par un service à la demande ou même en IaaS, la question se posera moins. Vous devez faire un cluster Hadoop, en général, ce sera avec une base Linux.
Requêtage, modèle de programmation, API	Tout type	Pour utiliser ou injecter les données, le code et les couches techniques ne sont jamais loin. Comment les données seront-elles interrogées ? Ai-je besoin d'écrire et d'exécuter des requêtes complexes ? SQL est-il indispensable (si oui, attention aux bases NoSQL) ? Quels langages sont supportés par défaut ? En Big Data, nous allons vite nous retrouver avec Java et R (dans certains usages très précis) ou du Python. C# sera par exemple limité à quelques bases.
Disponibilité de développeurs / DBA / compétences	Tout type	Un point à ne jamais négliger ! Est-ce que j'ai les compétences ou est-ce que je peux facilement trouver des compétences / profils sur la base choisie ? Faut-il me former ou former une équipe ? Quel coût ? Quel délai ? Si vous changez de solution de données, la question se posera.


Quentin Adam,
@waxzce, *CEO*
@ Clever Cloud


Laurent Doguin,
@ldoguin, *VP DevRel*
@ Clever Cloud


Ludovic Piot, @lpiot,
Lead of Customer Care
@ Clever Cloud

SQL vs. NoSQL : Data is eating the World!

C'est en paraphrasant Marc Andreesen(1), que je veux introduire cet article, et vous faire prendre conscience, s'il était besoin, de la place prépondérante qu'occupe la donnée numérique aujourd'hui. Et donc, ses technologies de stockage et de gestion.

S'il s'agissait déjà d'un enjeu voilà 15 ans, alors que j'étais un *rookie* de l'IT, c'était essentiellement pour des questions de disponibilité et de qualité de la donnée. Oracle régnait en maître incontesté dans les DSI. Les DBA en représentaient l'aristocratie enviable. Et les principaux points de vigilance étaient de déterminer la *source of truth* de la donnée et de s'assurer que toute donnée dérivée ou en relation était cohérente avec cette donnée de référence. On parlait alors d'ETL. Et les stockages de plus de 100 Go de données se trouvaient surtout dans les *datawarehouses* ; on y déversait la donnée dans des *datamodels* optimisés pour que l'on puisse les triturer selon tous les axes pour en tirer tout l'intérêt métier.

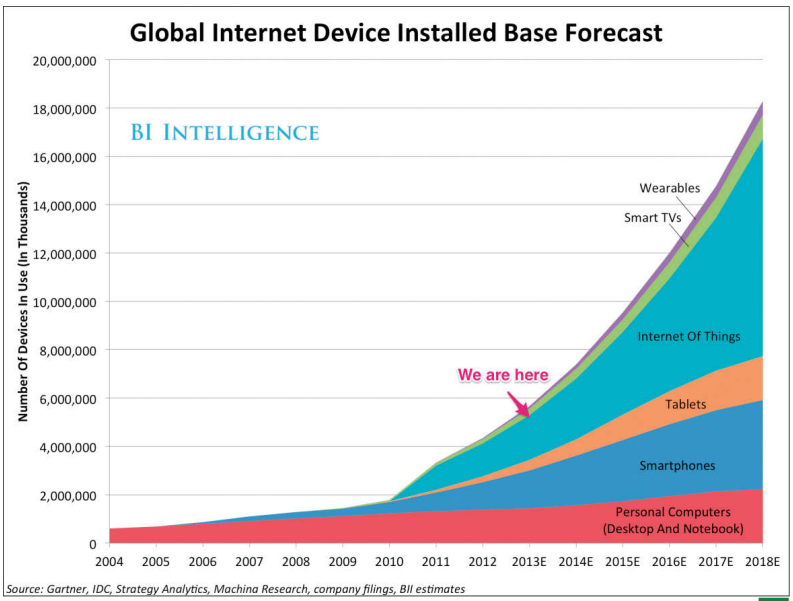
Mais depuis cette époque, *Steve Jobs* a lancé l'iPhone, bouleversant une fois encore l'écosystème tech en introduisant une nouvelle forme de consommation d'Internet en mobilité. Les *smartphones* et leurs applications mobiles ont démultiplié la nature et l'usage des données numériques.

A tel point que certaines des plus grandes réussites économiques de ces dernières années reposent quasi-exclusivement sur la capacité à gérer efficacement de la data : Uber, plus grande compagnie de taxi au monde… ne fait rouler aucun taxi en propre. Facebook, plus grosse plateforme média au monde… ne produit aucun contenu. AirBnb, très gros prestataire hôtelier… ne dispose d'aucun bien immobilier.

SoLoMo & IoT

Un acronyme décrit ces usages liés *aux smartphones*, et aujourd'hui vieux de 10 ans : SoLoMo, pour **So**cial, ***Lo***cal et **Mo**bile. **So**cial, d'abord, dans la mesure où tout consommateur d'une application, d'un usage, devient acteur de cet usage et producteur de donnée. Je donne un avis sur un restaurant, un lieu de vacances, j'échange des commentaires, des informations, des photos et vidéos avec mes amis sur les réseaux sociaux. En février 2018, Facebook est utilisé chaque mois par 2,13 milliards d'utilisateurs, qui sont autant de producteurs de données en puissance.

Local, ensuite, parce qu'avec ses capteurs intégrés (et son GPS en tout premier lieu), le *smartphone* produit lui-même de la donnée renseignant sur les comportements de son utilisateur dans son environnement quotidien. *Waze* en est l'exemple le plus illustratif : déduisant l'existence des bouchons par recoupement des données GPS des smartphones en déplacement.



1

Mobile, enfin, parce que les usages en mobilité réclament un accès instantané à la donnée, et à la donnée fraîche. Pour reprendre l'exemple de *Waze*, si l'application m'alerte de l'existence d'un bouchon quand je suis déjà dedans, c'est loupé. De même, si elle m'informe d'un bouchon qui s'est dilué depuis 30 minutes déjà. La pertinence de l'application va être remise en question par son utilisateur… qui ne manquera pas d'en faire part sur les réseaux sociaux.

Des capteurs, des applications, une certaine autonomie de fonctionnement, une connexion à Internet : à bien y regarder, le *smartphone* n'est que le premier représentant d'une foultitude foisonnante d'objets électroniques produisant de la donnée sur l'environnement réel de leur propriétaire, l'***IoT*** (***I***nternet **o**f ***T***hings). Cette estimation du Gartner illustre bien la croissance exponentielle du nombre de ces objets connectés, qui sont autant de producteurs de données en puissance. 1

Data… data everywhere!

A la lumière de ce graphique, on comprend assez vite que SoLoMo fait émerger de nouveaux enjeux dans la gestion de la donnée numérique. Ces enjeux ont été référencés par Gartner selon 5 dimensions.

(1) En 2011, cet entrepreneur et investisseur, au board de HP écrivait un essai dans le Wall Street Journal fondateur de la transformation digitale, expliquant pourquoi toute société de l'économie réelle devait devenir une société du numérique, il est également fondateur de netscape communication.

Volume

Des enjeux de stockage tout d'abord : entre 2013 et 2015, plus de données ont été produites que depuis la naissance de l'humanité. Et d'ici 2020, nos données occuperont pas moins de 44 Zo(2).

Variété

Des enjeux de format de stockage, car la multiplication des sources de production de données conduit à une explosion de la part de données non structurées, passant de 20% en 2008 à 87% en 2020(3).

Vélocité

Des enjeux de capacité de traitement de cette masse de données, également. D'abord au niveau vitesse : en 2020, chaque humain produira 2 Mo de données par seconde, qu'il faudra traiter, réconcilier, agréger et stocker. De même, difficile pour l'utilisateur moyen de patienter plus de 3 secondes avant d'avoir un résultat de recherche, l'état du trafic ou de démarrer le visionnage de la dernière vidéo Netflix.

Mais aussi au niveau de la capacité d'adresser l'ensemble de cette masse colossale de données au sein de traitements quasi temps réel. Aujourd'hui, moins de 30% des données produites et stockées sont exploitées. Chaque requête Google produit le résultat agrégé du travail de plus de 1000 serveurs en 0,2 sec !

Véracité et Visibilité

Enfin, des enjeux de résilience, de cohérence et de disponibilité de la donnée : inutile de dire qu'avec une telle pression sur la production et la consommation des données, on attend des bases de données qu'elles disposent de fonctionnalités de vérification de l'intégrité des données, de leur réplication à travers la planète, et de leur disponibilité malgré toutes sortes de panne potentielles.

Maintenant que ces enjeux sont posés, voyons comment l'industrie IT répond à ces problématiques et quelles sont les avantages et les limites des solutions technologiques proposées.

L'émergence du modèle NoSQL

Jusqu'au début des années 2000, on avait donc le choix entre Oracle, MS SQL Server, MySQL, IBM DB2 et Sybase ASE. Autrement dit, entre de la base de données relationnelle, ou de la base de données relationnelle.

Et si des différences sérieuses existaient en matière de coût de licence, de robustesse transactionnelle (*myISAM(4)*, si tu nous regardes !), de fonctionnalités avancées de requêtage ou d'administration, on restait dans les mêmes paradigmes d'architecture.

La donnée structurée

Pour manipuler les données de manière systématique, elles sont écrites suivant un modèle structuré défini avant usage, optimisant les requêtes en lecture par l'usage d'index statiques validant la qualité de la donnée par le respect de contraintes (unicité, respect du format, etc.).

Des contraintes relationnelles valident chaque transaction avant écriture. En évitant la redondance des données à gérer, on limite les écritures et le volume de stockage nécessaire, on accroît ainsi la performance en écriture, ainsi qu'en lecture (le cache en RAM présentant un meilleur *hit ratio*). On délègue surtout au niveau du moteur de base de données la garantie de la cohérence de la donnée : quelle que soit l'adresse des clients dans la ville de *Conflans-Sainte-Honorine*, l'orthographe de la ville est systématiquement identique (et *hopefully* correcte !).

Ainsi donc, pour un usage donné, on va optimiser les requêtes de lecture en faisant le design d'un modèle de données pertinent. Et l'on s'assure, à chaque écriture, de conformer la donnée entrante à ce format. C'est idéal… pour peu qu'on connaisse en avance de phase le format de la donnée que l'on va avoir à manipuler, la pérennité de ce format dans le temps, et aussi l'usage que l'on va faire de cette donnée dans le temps.

C'est ainsi que, dans les années 2000, les projets de *datawarehouse* / *datamart* fleurissaient. Ils consistaient à produire des dépôts de données présentant des modélisations multiples, chacune étant optimisée pour répondre le plus efficacement à un type de requête complexe, multiaxiale.

Single-node… ou presque

Cette structuration des données répondait à une préoccupation : malgré l'abstraction que présente SQL, il fallait tirer la meilleure performance du *hardware* sous-jacent. Car dans cette architecture monolithique, le serveur de base de données est seul à assumer les écritures de données, la gestion des connexions, des transactions et la disponibilité de la donnée. Bien sûr, des nœuds *slaves* sont venus délester le serveur principal de la pression des requêtes en lecture. Il n'en reste pas moins que cette architecture établit une attraction gravitationnelle sur le *master node* que l'on bichonne(5) en lui offrant un hardware gérant la résilience aux pannes matérielles(6) et donc très coûteux.

Pourtant, cette conception de la résilience n'est assurée qu'au niveau local et ne résiste pas à la perte d'un rack, d'un équipement cœur de réseau ou d'un datacenter.

SQL et ACIDité

Reste que malgré ces contraintes fortes, les moteurs de base de données relationnelles présentent 2 atouts essentiels qui, jusqu'à aujourd'hui continuent d'assurer la pertinence des bases de données relationnelles, malgré les limitations décrites précédemment.

SQL, tout d'abord. Ce DSL de requêtage de données universel, standardisé(7) reste extrêmement pertinent. Il s'appuie sur l'algèbre des ensembles, pour abstraire la manière dont le moteur de SGBD-R va aller chercher la donnée : une requête se conformant à retourner toutes les données d'une table (et optionnellement ses jointures) répondant à des critères donnés, avec ou sans agrégation. Ce langage présente une concision, une expressivité et une efficacité telles que bon nombre de technologies concurrentes tentent (avec plus ou moins de bonheur) de singer son comportement, sinon de s'en inspirer fortement. Je pense à l'*OQL* de *Gemfire*, ou plus récemment au *CQL* de *Cassandra*.

---

(2) 44 Zetta octets = 44.1012 Go. Source Forbes
(3) Source IDC : The Digital Universe 2014
(4) http://sql.sh/1548-mysql-innodb-myisam
(5) le fameux pet du cattle vs. pet
(6) alimentation redondée, mémoire ECC, disques en RAID matériel avec cache sauvegardé sur batterie, composants remplaçables à chaud.
(7) ANSI-82 et ISO-83, puis des révisions régulières jusqu'en 2016.

Sur un composant central d'accès à la donnée, la gestion de la concurrence d'accès est un élément critique pour garantir la cohérence des données : si, quand une transaction de carte bancaire est annulée, le compte de son propriétaire est, malgré tout, débité, ça va un tout petit peu remettre en cause les fondamentaux de l'économie mondiale :-). Là encore, la gestion des propriétés *ACID*(8) d'une transaction d'écriture de données est un mécanisme robuste, éprouvé par des décennies d'usage que les SGBD-R portent nativement, évitant au développeur d'avoir à se charger de réinventer ces mécanismes relativement complexes et absolument critiques.

Les principes fondateurs du NoSQL

Pourtant, au début des années 2000, Google fait émerger le *NoSQL* et la *Big Data*, à travers deux articles fondateurs, devenus célèbres depuis : Google File System(9) et MapReduce: Simplified Data Processing on Large Clusters(10).

Architecture distribuée

Ils partent du constat que même le plus gros, le plus puissant et le plus coûteux des serveurs de base de données traditionnel n'est plus suffisant pour adresser le volume de données que l'on souhaite gérer. Autrement dit, les capacités d'*upscaling* sont atteintes.

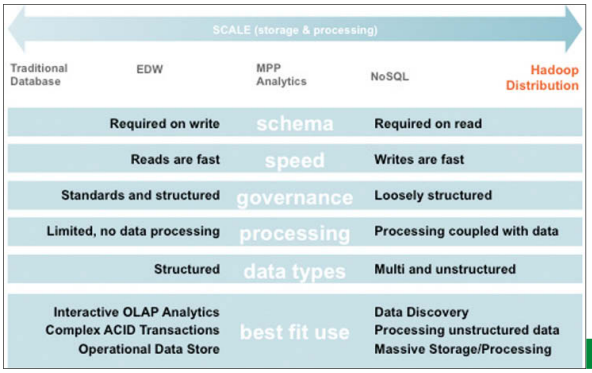
Un constat connexe est que la *"haute disponibilité"* d'une telle architecture reste toute relative, limitée à la résilience *hardware* proposée par l'onéreuse baie *SAN* et le *cluster* master/slave de gros serveurs départementaux. Cette haute disponibilité n'est plus suffisante pour adresser des usages grand public sur le Web, adressant des millions de visiteurs uniques à travers les 24 fuseaux horaires et les 5 continents.

Pour répondre à ces besoins, Google a fait le pari d'une architecture x86 classique, mais massivement distribuée avec des mécanismes logiciels de distribution, de réplication et de convergence de données complètement distribués.

Cette architecture repose énormément sur le réseau, bien plus lent que les disques. Pour conserver une performance acceptable malgré le handicap de la lenteur du réseau, Google a inventé le *pattern* de développement *Map-Reduce*, qui consiste à renverser la logique de transit. Plutôt que faire transiter la masse des données brutes jusqu'aux nœuds de traitement, on va distribuer le code logiciel de traitement des données (bien plus léger) directement sur les nombreux nœuds qui portent la donnée (étape du *Map*), puis à rapatrier les résultats intermédiaires (moins volumineux) de ces traitements sur des nœuds de consolidation (étapes de *Shuffle* et *Reduce*). Ainsi, on limite le transit des données sur le réseau, et on démultiplie le traitement en parallèle sur l'ensemble des nœuds portant des données nécessaires au traitement.

En poussant ce principe de distribution plus loin, aucun nœud n'est plus un SPOF(11), et l'on obtient un système de gestion des données qui voit ses capacités de stockage et de traitement qui croissent de manière linéaire, là où les SGBD-R s'essoufflent bien plus vite.

Les clients peuvent envoyer leurs requêtes sur un pool de nœuds,



2

qui vont eux-mêmes aller requêter les nœuds de données à travers des mécanismes d'orchestration eux-mêmes distribués.

Si plus aucun nœud n'est un SPOF, la résilience du système peut, elle aussi, augmenter linéairement avec le nombre de nœuds répliquant la même donnée. La topologie de ces nœuds pouvant s'étendre au-delà du serveur, du rack, du datacenter, du pays, du continent.

Donnée non structurée

Ces principes de distribution des données et des traitements sur de l'infrastructure bon marché conduit à une libération de la créativité dans la manière de gérer la donnée. Puisque la puissance brute de traitement n'est plus un (gros) problème, puisque le stockage est disponible au coût du simple disque dur S-ATA, alors la redondance des données est envisageable, et requêter des données hétérogènes à partir de formats non structurés devient possible : consolider de la donnée entre un fichier de log et un fichier *JSON* avec un pivot au format CSV n'est plus aberrant, mais au contraire présente une certaine logique : celle de limiter au maximum les transformations au moment de l'écriture, et laisser la capacité de traitement distribuée de la lecture aux multiples nœuds de calcul. 2

SUPER DATA MAN A TOUJOURS PORTÉ UNE CAPE

Pour étendre ce qui a été vu au-dessus, on peut en déduire que chaque solution de stockage de données a ses intérêts et ses inconvénients. Ça a même été théorisé, et ça se nomme le *CAP Theorem*. ***CAP*** veut dire :

Consistency, ou l'homogénéité, ça définit la capacité du système à garantir que la donnée qu'on lui confie est uniformément accessible, que les mises à jour de la donnée sont propagées entre chaque requête. C'est sur cette caractéristique que reposent les transactions.

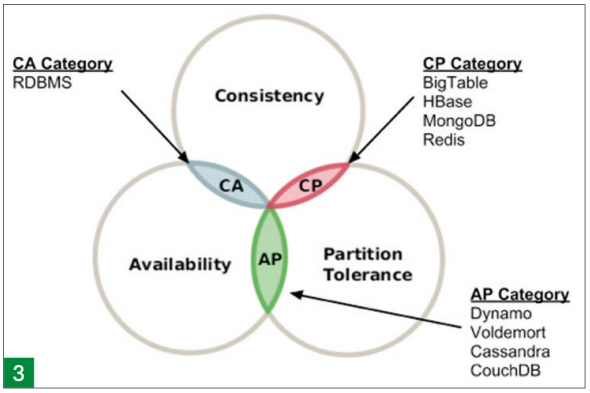
Availability, la disponibilité, définit la résilience d'accès d'un système, généralement basé sur la copie de la donnée au sein d'un cluster, c'est sur cette caractéristique que l'on construit les modèles de haute disponibilité, pour des systèmes qui ne peuvent pas supporter l'indisponibilité.

Partition tolerance, la capacité à gérer la panne partielle, cette capacité est ce qui permet de reconstruire un *cluster* en cas de désynchronisation réseau. Mais aussi de savoir quelle partie d'un *cluster* va rester active si la partition réseau vient à scinder le *cluster* en morceaux n'ayant plus de communication les uns avec les autres.

De ces trois caractéristiques, le *CAP theorem* explique qu'on ne

(8) https://fr.wikipedia.org/wiki/Propri%C3%A9t%C3%A9s_ACID
(9) https://research.google.com/archive/gfs.html
(10) https://research.google.com/archive/mapreduce.html
(11) Single Point of Failure : ou maillon faible

peut jamais bénéficier des trois à leur plein potentiel, que l'on peut en pousser deux en même temps, mais jamais trois. Et qu'ainsi chaque système de données doit faire des choix basés sur ses contraintes et privilégier ses besoins au détriment de ce qui n'est pas obligatoire. Il faut également prendre en compte les deux contraintes supplémentaires. Le stockage, car en cas de volumétrie forte de données, il faudra obligatoirement partager cette donnée entre plusieurs instances, en la découpant, ce qui compromet l'utilisation des *mono nodes*. Ensuite, les performances, qui, si elles sont critiques peuvent fortement impacter les choix. En effet, la synchronisation systématique avec réconciliation des données est lente et coûteuse en performances. 3



3

Dans la pratique, *challenger* les capacités d'une technologie en regard du *CAP theorem* n'est pas chose aisée. Bien souvent les éditeurs ou les communautés ont une tendance à l'emphase et à l'optimisme quant aux performances de leur poulain. Au-delà du test empirique et artisanal, un homme s'est dressé, dès 2013. Kyle Kingsbury *a.k.a Aphyr*(12) a développé un outil de benchmark pour système de persistance de données distribué nommé Jepsen, et publie régulièrement des analyses poussées(13) de ses résultats. Une mine d'or pour tout développeur qui doit choisir entre l'outil *hype* mais très en-deçà de ses promesses *marketing* et l'outil *rock-solid* mais pas sexy pour un sou. Au-delà, *Aphyr*, par ses blog posts et ses interventions, a énormément contribué à faire évoluer les bases *noSQL* (et les bases distribuées en général), en poussant les éditeurs à revoir leur copie. Ainsi, le dernier *blog post* concernant *MongoDB* a été sponsorisé par l'éditeur, en guise de droit de réponse à 2 analyses antérieures aux résultats plus que passables. Entre temps une série de patches était passée par là ! ;-)

Eventual consistency

Ce *CAP theorem* est un sérieux caillou dans la chaussure des fournisseurs de solutions *NoSQL* : pour adresser beaucoup de données, il faut distribuer charge de traitement et stockage. Mais ce faisant on introduit avec l'infrastructure réseau une latence et une faiblesse qui contreviennent à la cohérence des données. La notion de transaction et les propriétés ACID sont mises à mal.

C'est en remontant aux enjeux métiers que Google se rend compte que… la cohérence des données n'a pas forcément besoin d'être instantanée.

Si c'était vrai dans l'univers de la base relationnelle, où une donnée sert à chaque fois que l'on en a besoin, ça ne l'est plus forcément dans l'univers déstructuré du NoSQL, où chaque parcours d'achat peut vivre avec ses propres données. Peu importe qu'un panier d'achat d'un utilisateur doive être livré à *"Conflans St Honorine"* et qu'un autre soit livré à *"conflans-sainte-honorine"*, car chaque panier va stocker sa propre adresse de livraison de manière dissociée. Et les 2 valeurs d'adresse restent valides au regard du métier logistique.

On pourra toujours passer un batch toutes les heures qui s'occupent de "nettoyer" les différentes adresses pour retrouver une certaine conformité avec un référentiel. Ou alors c'est lors de la production des *dashboards* de *reporting* métier qu'on fera cette consolidation.

On remplace une ACIDité technique, par une cohérence fonctionnelle plus applicative.

Sur le plan technique, le report du changement de la donnée sur tous les nœuds porteurs de cette donnée peut se faire en quelques secondes de manière asynchrone, sans que l'impact soit énorme sur le fonctionnel.

Ainsi, le stock d'un produit peut très bien être mis à jour de manière asynchrone, par rapport aux actes d'achat, par l'appel d'un service applicatif dédié. Si les quelques secondes/minutes de décalage amènent au cas limite où le stock ne suffit pas à honorer les commandes passées pendant ces quelques secondes, on peut toujours gérer le cas au niveau fonctionnel, par la notification d'un retard de livraison, par exemple.

Bienvenue à la *"cohérence à terme"*, où l'on apprend à vivre avec la faiblesse technique en palliant par des astuces au niveau de la gestion applicative. Et c'est là une différence d'approche fondamentale entre SQL et NoSQL : pour pallier aux contraintes techniques posées par les SGBD-R, les bases NoSQL s'émancipent de certaines fonctions de gestion de la donnée, qu'il faudra donc adresser au niveau applicatif, voire métier.

On parle alors de *BASE* pour *Basic Availability, Soft State, Eventually Consistent* en opposition au terme *ACID*.

Basically Available : la base de données peut toujours répondre à une requête, cf. le *CAP theorem*.

Soft State : l'état du système peut changer au cours du temps, ce qui est dû à l'*Eventual Consistency*.

Eventually Consistent : la cohérence de la donnée n'est pas toujours garantie.

LES DIFFÉRENTS NOSQL

Vous l'aurez compris, les bases NoSQL se sont dès le début opposées aux bases *"SQL"*. *SQL* entre guillemets puisqu'en réalité il s'agit plus d'une opposition au modèle traditionnel de SGBD-R et sa difficulté à passer à l'échelle. Mais SQL est un abus de langage facile pour parler de SGBD-R, le terme NoSQL y gagnera donc en flou artistique. Pour décrire les différents types de base NoSQL, regardons les non-SGBD-R.

(12) https://aphyr.com/about
(13) https://jepsen.io/analyses

Le Clé-Valeur

Il est intéressant de commencer avec la *clé-valeur*, en anglais *KV store*. Un nom très intéressant puisqu'il ne contient pas de référence à la base de données. Et pour une bonne raison puisque l'on ne peut pas faire de requête sur un *KV store*. Il est seulement possible de stocker ou récupérer un couple clé-valeur. Si vous pouvez récupérer une valeur autrement que par sa clé, alors votre *KV store* a évolué en base de données.

Quelque exemples :

clé	valeur
userSession99	Q2VjaSBlc3QgdW4gY291cXZlciBDbG91ZDogUFJPR1JBTU1FWg==
User99_username	'ldoguin'
User99_age	32

Memcached est un de ses représentants les plus utilisés encore aujourd'hui. C'est un *KV store in-memory*. Toutes les paires clé-valeur sont stockées en mémoire vive. Son principal cas d'utilisation est donc le cache. La première version de *Memcached* n'était pas distribuée. Elle a évolué pour donner *Membase*. Cette nouvelle version proposait le support de la persistance, de la réplication et une gestion simple de *clustering*. Il était notamment utilisé par *Facebook*, *Zynga*…

On pourra citer d'autres produits comme *Redis*, *Riak*, *Hazelcast*, *EHCache* , *ZooKeeper*, *etcd*, *Consul*, et quantité de solutions plus récentes comme *LevelDB*, *RocksDB*, *BoltDB* et beaucoup d'autres.

Beaucoup ont évolué en ajoutant la possibilité de créer des index et de les requêter, les transformant ainsi en base de données. Nous allons en détailler certains modèles.

Base orientée Document

Une base orientée Document permet de récupérer une ou plusieurs paires clé-valeur en fonction de la partie valeur. Elles sont généralement stockées sous forme de *XML* ou *JSON* mais peuvent aussi être atomiques. On peut les retrouver directement dans un *KV store* ou en valeur de colonnes de certaines bases de données. Le principal intérêt de ce modèle est l'absence de schéma AVEC la possibilité de les requêter. On parle alors de base *Schemaless*. D'ou la question suivante : contre quels champs effectuer sa requête ?

C'est tout l'intérêt d'utiliser *XML* ou *JSON*, puisqu'en fait ils permettent de faire du semi-structuré. Le schéma n'est pas défini au niveau de la base mais au niveau du document. Quelque part vous pouvez avoir autant de schémas que vous voulez et donc le changer quand vous voulez. Cela apporte une grande flexibilité, plus besoin de migrer toute une base d'un seul coup quand par exemple on change son schéma. L'avantage n'est pas de pouvoir faire absolument n'importe quoi mais d'avoir de la flexibilité, d'avoir la possibilité de changer au moment opportun. Vous en lirez plus sur ce sujet dans la partie modélisation de données.

Quelques exemples :

```
user99 {"username":"ldoguin","age":32}
user99 <user><username>ldoguin</username><age>32</age></user>
user_counter 100
```

La plus connu est certainement *MongoDB*. On trouvera aussi *CouchDB*, *Couchbase*, *Aerospike* entre autres.

Base orientée Colonnes

Conceptuellement les plus proches des SGBD-R traditionnels, les bases colonnes permettent de stocker des informations par colonnes et non par lignes. Prenons l'exemple suivant :

Stockage en lignes :	Stockage en colonnes :
097,97,lpiot,null;	97:097,98:098,99:099
098:98,waxzce,29;	lpiot:097,Waxzce:098,doguin:099;
099:99;,doguin,32;	29:098,32:099;

Le stockage en colonnes ressemble un peu à une succession d'index. Chaque ligne représente en quelque sorte un index sur l'identifiant, le nom, puis l'âge. On peut voir qu'un des premiers avantages est la compression. Si l'âge de Ludovic n'est pas renseigné, il est *null* dans un stockage par lignes et simplement omis dans un stockage par colonne.

Selon le *use case* cela représente un gain de place non négligeable. Un index plus petit sera parcouru plus vite. Autre avantage en termes de performance, puisque toutes les entrées d'une ligne sont du même type, il est facile d'appliquer un algorithme de compression pour réduire encore plus l'espace occupé.

De fait les bases colonnes sont souvent utilisées pour des charges analytiques sur des données de type similaire. On retrouvera des cas d'utilisation dans la statistique, la *business intelligence*, le *machine learning* ou encore le *monitoring*.

Parmi les bases colonnes connues on mentionnera *Cassandra*, *BigTable*, *HBase* ou encore *Vertica*.

Base timeseries

Les bases *timeseries* ou (TSDB pour *time series database*) peuvent être vues comme une spécialisation des bases colonnes. Elles sont faites pour stocker des séries temporelles. On assiste en ce moment à un renouveau du secteur pour deux raisons principales.

D'abord l'ubiquité des objets connectés, ceux-ci doivent en effet écrire quelque part, et ce qu'ils écrivent, en général, correspond bien à une modélisation *timeseries*. On peut prendre comme exemple des capteurs de métriques (température, vitesse du vent, rythme cardiaque, glycémie…).

L'autre raison nous vient de l'informatique distribuée et du *cloud*. Plus on a d'applications dans son architecture et plus notre besoin de les monitorer augmente. La mode allant progressivement dans le déploiement d'architectures clé-en-main avec des orchestrateurs comme *Kubernetes*, il fallait bien une brique dédiée au *monitoring*.

On a donc vu apparaître de nouvelles bases comme *Prometheus*, *InfluxDB*, *Graphite*, *RiakTS*, ou encore *Warp10*. Ce dernier est utilisé par Clever Cloud et OVH pour gérer leur infrastructure de *monitoring*. Venez donc nous rejoindre en meetup(14) !

Base orientée Graphe

Pour les bases de données graphes, le changement de paradigme est plus visible. Ici, les données sont stockées sous forme de triplets. Ils sont constitués d'un arc reliant deux nœuds. On pourrait, dans le cas d'un réseau social, avoir comme nœuds deux différentes personnes reliées par un arc représentant une relation entre ces deux

(14) https://www.meetup.com/fr-FR/Warp10/

personnes. Chantal est amie avec Alain par exemple. Ceci peut être facilement exprimé en *SQL* après tout. On pourrait avoir un champ tableau contenant la liste des identifiants des amis.
Alors pourquoi une base graphe ? On commence à s'amuser quand il faut savoir qui sont les amis des amis des amis de Chantal. Imaginez comment trouver ce résultat avec une base *SQL* et les jointures à faire. Une base graphe répondra beaucoup plus facilement à ce besoin. Pour ce faire elle utilisera une autre structure de données ainsi qu'un autre langage que *SQL*. Dans le cas de *Neo4J* il faut utiliser *Cypher*. Voici un exemple pour vous donner une idée de comment exprimer une requête graphe :

```
CREATE (c:User { name: "Chantal" }),
(a:User { name: "Alain" }),
(d:User { name: "Dominique" }),
(c)-[:KNOWS]->(a),
(c)-[:KNOWS]->(d),
(a)-[:KNOWS]->(c),
(a)-[:KNOWS]->(d),
(d)-[:KNOWS]->(a),
(d)-[:KNOWS]->(c)

MATCH (u:User)-[:KNOWS]-(ami)
RETURN u, ami
```

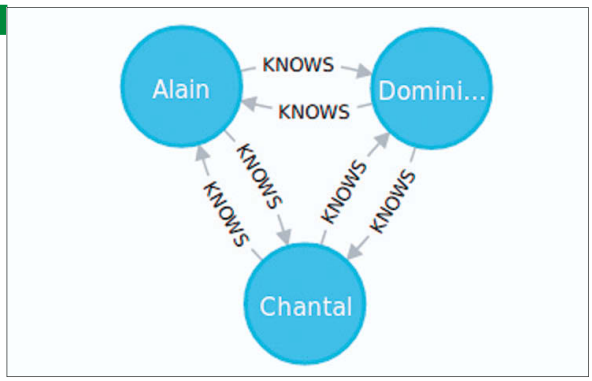
4

Parmi les différentes bases graphes on mentionnera *Neo4J*, *AllegroGraph* ou encore *InfiniteGraph*. Certaines bases proposent aussi un support Graphe. En fait la plupart des nouvelle bases proposent une manière de requêter un graphe. C'est le moment d'aborder les base multi-modèles.

Les bases multi-modèles

La popularité et la banalisation du *NoSQL* a permis de s'ouvrir à de nouveaux modèles et donc de prendre conscience de leur meilleure adéquation pour tel ou tel cas d'utilisation. En prenant aussi en compte la popularité croissante du modèle *micro-services*, on se retrouve assez vite à faire cohabiter plusieurs bases aux modèles différents et complémentaires dans son architecture. Est-il pour autant raisonnable d'administrer une base par modèle ? Ne pourrions-nous pas utiliser une base qui propose à la fois du *KV*, du document, du relationnel et du graphe ?
Ainsi nous avons vu apparaître de nouvelles bases dites multi-modèles et certaines *"anciennes"* base *NoSQL* ajouter le support de

4



nouveaux modèles pour rester dans la course. Elles proposent généralement une façon de stocker les données et plusieurs façons de les indexer.
On notera parmi les plus connues *ArangoDB*, *OrientDB*, *Couchbase* ou même *Oracle*.

LA MODÉLISATION DES DONNÉES

Le design de clés

On peut identifier trois types de clés : déterministe, généré automatiquement et composé des deux précédents.

```
"user::99":{
   "username":"ldoguin",
   "telephone":"0000000000",
   "email":"laurent.doguin@clever-cloud.com",
   "age":32,
   "orders":[2,5,7,10]
  }
```

Clé déterministe

Rappelons-nous que dans le cas d'un *KV*, on ne peut récupérer une valeur qu'en connaissant sa clé. Il est donc important d'utiliser une clé déterministe. Plusieurs *patterns* sont possibles. Prenons comme exemple un objet Utilisateur. Lorsqu'il s'authentifie, l'utilisateur va utiliser comme *login* un élément dont il peut se souvenir. On utilisera alors une clé facile à se rappeler, par exemple son e-mail.
Que se passe-t'il quand l'utilisateur décide de modifier son e-mail ? Il faudrait mettre à jour toutes les références à ce profil utilisateur dans les autres documents. On pourrait se satisfaire de cette situation en se disant que cela ne demande pas beaucoup de travail. Ou alors on pourrait utiliser quelque chose que l'utilisateur ne changera pas.

Clé générée automatiquement

Pour éviter le problème mentionné précédemment on va utiliser un *UUID* généré aléatoirement ou éventuellement un compteur. Certains *KV stores* permettent même d'utiliser des compteurs atomiques. Mais alors comment fait-on pour retrouver un utilisateur si son identifiant est 99 ou encore 8b7a4320-44d0-45e1-9285-cc3db8f12ba6 ? S'il n'est pas possible de créer des index on utilisera des *lookups keys*.

Lookup Keys

Les *lookups keys* sont un *pattern* courant qui permet de simuler des index secondaires. Le principe est simple. Si l'on veut récupérer un profil utilisateur en utilisant son e-mail, il nous faut un index, en gros une table avec 2 colonnes. Une colonne avec l'e-mail de l'utilisateur et une avec la clé du profil. Cet index peut être remplacé par une nouvelle paire clé-valeur pour chaque ligne. Il faut donc créer deux paires clé-valeur lorsque l'on crée un profil. La première aura pour clé l'identifiant généré automatiquement et comme valeur le profil. La seconde aura pour clé l'e-mail du profil, et pour valeur l'identifiant du profil. On peut alors faire deux opérations successives pour retrouver un profil utilisateur via son e-mail.

Création d'un utilisateur

```
Client  Serveur
id = incrementCounter()
put(id,profile)
put(email,id)
```

Récupération du profil utilisateur par e-mail

```
Client  Serveur
id = get(email)
user = get(id)
```

Lorsque l'on vient du *SQL*, faire deux appels consécutifs peut sembler contre-intuitif. Nous avons pour habitude de réduire au minimum la sollicitation de notre base *SQL*. Même un simple *SELECT* passe par une quantité non négligeable de logique avant de répondre. Ces appels se chiffrent en général en millisecondes. Il est donc important de rappeler que la plupart des *KV stores* répondent sous la milliseconde pour de simples *get* (en supposant que votre jeu de données tienne en RAM). Il sera souvent plus efficace d'enchaîner plusieurs appels sur un *KV store* que de faire une seule requête avec jointure sur une base *SQL*. Il y a bien sûr un degré de complexité où la tendance s'inverse. Imaginez juste une requête *SQL* qui réponde en 10 ms. Pendant ce temps-là, vous avez fait au moins 10 appels *KV*. N'hésitez donc pas à utiliser des *lookups keys*. Gardez bien en tête que celles-ci sont souvent déterministes et peuvent donc être modifiées par l'utilisateur. Mais au moins, vous n'avez qu'une modification à faire cette fois.

Clé composée

Nous n'avons pas encore abordé le sujet de la dénormalisation. En *SQL*, on assume que tout votre modèle de données sera normalisé, que la donnée ne sera jamais dupliquée. Prenons pour exemple un objet commande de site e-commerce dans lequel on stockera tous les produits faisant partie de la commande. Une grossière modélisation *SQL* donnerait une table "produit" et une table "commande". La table "commande" aurait une colonne "produits" qui serait un tableau de clés étrangères référençant chacune une ligne de la table "produit".

```
"order::1" : {
 "products":"[1,4,6,9]"
}
```

Dans un modèle clé/valeur on pourrait le représenter comme ceci :

```
"order::1" : {
 "products":"[
 {"id":"1","price":66,"designation":"Hipster Beard Oil","image":"https://myimages.com/hbo.png"},
 {"id":"4","price":-30,"designation":"Flanel Shirt","image":"https://myimages.com/fs.png"},
 {"id":"6","price":0,"designation":"Jean Slim fit","image":"https://myimages.com/jsf.png"},
 {"id":"9","price":10,"designation":"Suspenders","image":"https://myimages.com/s.png"}]"
}
```

Ici, on embarque les documents produits directement dans la commande. L'avantage est que pour afficher une commande il faut uniquement récupérer l'objet "commande". En *SQL*, il faudrait faire une requête non triviale. Le désavantage est que si l'on veut mettre à jour l'URL d'une image il faut la mettre à jour partout, dans l'objet "produit" et dans les objets "commandes" qui embarquent ce produit.

Si l'on veut rester dans l'esprit clé-valeur on peut trouver un compromis en utilisant des clés composées, c'est-à-dire une clé avec une partie déterministe et une partie automatique. On pourrait créer une paire clé-valeur spécifiquement pour l'image d'un produit en connaissant sa clé. Si la clé du produit est p:1 alors on pourrait avoir une paire p:1::image ayant pour valeur "https://myimages.com/hbo.png". Ainsi nous n'avons plus le problème de cohérence de la donnée pour le champ "image". Nous pouvons récupérer la valeur de ce champ de façon déterministe dès lors que nous avons l'identifiant du produit.

Une bonne modélisation de données clé-valeur se fera en prenant en compte tous ces compromis. Gardons à l'esprit que dans beaucoup d'architectures cohabitent une base de donnée en tant que *source of truth*, et un *KV store*. Les contraintes et compromis changent du tout au tout lorsque le clé-valeur est utilisée comme cache et non comme *source of truth*.

Du clé-valeur à la base de données

Nous l'avons évoqué précédemment, dans le *NoSQL* il est facile de dénormaliser. Une tendance qui vient de l'impossibilité d'exécuter des requêtes en *KV* et surtout du cas d'utilisation cache prédominant. Tendance qui disparaît avec la plupart des base de données *NoSQL* qui permettent maintenant d'utiliser un ou plusieurs langages de requête souvent inspirés par le *SQL*. Si l'on prend *Couchbase* en exemple, sont apparus successivement les index *Map-Reduce* (hérité de *CouchDB*) puis *N1QL* (similaire au *SQL* mais pour les documents *JSON*) puis le support du *full text*.

Pouvoir utiliser un langage qui permet d'effectuer des jointures, donc de retourner un résultat comprenant plusieurs documents relaxe la contrainte de dénormalisation. Ici tous les langages ne sont pas égaux. Leur capacité dépend beaucoup de comment est stockée la donnée et de comment elle peut être indexée.

Les Index Map-Reduce

Commun dans le monde *Hadoop* et popularisé par *CouchDB*, le *Map-Reduce* permet de créer et de *sharder(15)* un index très simplement. Il faut créer une, voire deux fonctions, pour le mettre en place. La première fonction est le *Map*. Cette fonction sera exécutée contre tous les éléments de la base et permettra d'ajouter une ou plusieurs lignes dans un index. Et quand on a un index, on peut faire une *query*. Petit exemple tiré de *CouchDB* :

```
function(doc, meta) {
  if (doc.type == "user") {
    emit(doc.id, null);
  }
}
```

Ici on a une fonction qui prend un document et vérifie que ce document à un champ type égal à user. Si c'est le cas, on peut ajouter une

(15) Sharder consiste à découper en des sous-ensembles techniques ou fonctionnels. Dans le monde SQL, le pattern du sharding est beaucoup utilisé pour repousser les limites d'un serveur : en découpant la base de données en bases plus petites, dédiées par zones géographiques, par exemple.

entrée dans l'index avec la fonction emit. On se retrouve alors avec l'équivalent d'une table à 2 colonnes avec l'ID du document en clé et null en valeur. Pourquoi null en valeur ? Ici on se servirait de cet index uniquement pour compter ou récupérer la liste des utilisateurs. On ne mettra donc pas le document en valeur par souci d'économie de place. On pourrait mettre une valeur simple comme l'âge de l'utilisateur ou encore un tableau avec seulement les valeurs que l'on souhaite afficher. L'important est d'avoir un index à la fin.

Et si on a un index, on peut faire une requête. L'intérêt du *Map-Reduce* est qu'il est très simple à distribuer. De fait si vous avez des données réparties sur 4 machines sur lesquelles vous faites tourner un *Map-Reduce*, vous avez un index partagé entre ces 4 machines. Un point fondamental à garder en tête est que, de fait, votre requête sera distribuée sur ces 4 machines. Il faut alors attendre le résultat de la requête pour chaque machine, les agréger puis les restituer. Un processus qu'on retrouvera souvent sous la dénomination *Scatter and Gather* dans la littérature anglophone. Ce processus est bien sûr plus long que d'interroger un index sur une seule machine.

Nous n'avons pas encore parlé de la partie *Reduce*. C'est une autre fonction, qui prend en paramètre le résultat de la fonction *Map* sur tous les objets de la base. On pourrait, dans notre cas, s'en servir pour compter le nombre d'utilisateurs présents. Voici un exemple simpliste :

```
function(key, values) { return values.length; }
```

Dans un cas plus concret, par exemple avec *Couchbase*, le *Map-Reduce* est auto-incrémental. On ne repasse pas les fonctions, sur chaque document, à chaque mutation de la base pour d'évidentes raisons de performance. Il faut donc pouvoir gérer ce cas d'auto-incrément. Dans le cas du *Map*, on va simplement passer la fonction sur les documents touchés. Dans le cas du *Reduce*, on va s'occuper du *delta*. Regardez l'exemple suivant :

```
function(key, values, rereduce) { if (rereduce) { var result = 0; for (var i = 0; i <
values.length; i++) { result += values[i]; } return result; } else { return values.length; } }
```

Ici le *Reduce* permet d'additionner les valeurs du nouveau *Reduce* (celui calculé uniquement avec le *delta*) à l'ancien. Il permet aussi de *scaler* plus facilement en utilisant les résultats de *Reduce* de chaque nœud ou en supportant l'auto-incrément.

Vous trouverez de nombreux exemples vous expliquant tout le potentiel du *Map-Reduce*. Ce n'est pas aussi intuitif que du *SQL* mais permet tout de même beaucoup de choses intéressantes. Si vous êtes curieux, je vous invite à regarder ce qui se fait pour *PouchDB*. Une surcouche aux bases embarquées de vos navigateurs, qui fonctionne avec du *Map-Reduce*. Certains *plug-ins PouchDB* vous permettent, par exemple, de faire du relationnel en utilisant du *Map-Reduce* et un nommage de clés bien particulier.

Gros bémol cependant pour les habitués du *SQL* avec le *Map-Reduce*. Il faut d'abord créer un index avant d'effectuer une requête.

Requête Ad-Hoc en NoSQL

On rentre dans une nouvelle catégorie de bases *NoSQL* puisque celle-ci ne nécessite pas la création d'index au préalable. *MongoDB*, *Couchbase*, *Cassandra*, *Neo4J* et beaucoup d'autres le permettent. Tous les différents langages de requête ne sont pas *iso* au *SQL* et n'ont pas la même expressivité. Sans rentrer dans les détails il faut faire attention à ce paramètre pendant ses choix de modélisations. Il est important de savoir si l'on pourra tout requêter en normalisant toute sa donnée. L'exemple qui me vient en tête est le support du JOIN dans *Cassandra*, qui pendant longtemps n'était possible qu'avec une solution tierce comme Spark. Ceci étant dû au modèle colonnes et aux choix de cohérence effectués.

En effet, dans un *cluster Cassandra*, on peut choisir de répliquer la donnée plusieurs fois. On pourra alors mettre à jour une valeur sur une machine et effectuer une lecture d'un *réplica* sur une autre machine avant que celui-ci soit mis à jour. S'ensuit une incohérence de la donnée. On peut bien sûr demander à tous ou certains des nœuds de se mettre d'accord. C'est ce qu'on appelle le *quorum* dans *Cassandra*. Bien sûr, se mettre d'accord prend plus de temps. On peut donc privilégier ici la disponibilité à la cohérence et *vice-versa*. Tout ceci se retrouverait fortement accentué si vous effectuiez des JOIN. On retrouve un peu la notion de *Scatter and Gather* évoquée précédemment. C'est une des raisons pour laquelle *Cassandra* décourage l'emploi de JOIN par défaut.

Pour revenir au sujet de la modélisation, il est important de garder en tête les limites du système de requête utilisé pendant le *design* de son modèle.

To Normalise or to Denormalise

Nous avons abordé ce point plusieurs fois dans les paragraphes précédents. En *NoSQL*, on choisit souvent de normaliser ou dénormaliser la donnée. Sur quels critères faut-il se baser pour faire son choix ?

Dénormaliser

Inconvénients

- Incohérence de la donnée : si la donnée est dénormalisée, dupliquée, alors il peut y avoir une incohérence entre ses versions. Soit cette incohérence est acceptable, soit il faut mettre à jour toutes ses versions ;
- Requêtage : les incohérences peuvent aussi se retrouver dans les requêtes. Comment s'assurer que l'on est bien en train de sélectionner ou de discriminer la bonne valeur ?
- Taille : tlus on dénormalise, plus on duplique, plus la donnée occupe de la place.

Bénéfices

- Rapidité : si l'on a toutes les données dans le bon document, la requête sera forcément plus efficace que s'il faut effectuer une jointure ;
- Tolérance à l'indisponibilité : tous les systèmes distribués peuvent souffrir de l'indisponibilité d'un de leur éléments. Si l'on perd le nœud d'un *cluster* contenant la fiche produit d'une commande, mais que l'on a embarqué le produit dans le document commande alors nous n'avons pas de problème.

Pour résumer il semble sage de dénormaliser lorsque :

- On privilégie la lecture à l'écriture ;
- On est tolérant au risque d'incohérence des données ;
- On privilégie la vitesse à la cohérence ;
- Le fait de dupliquer de la donnée est un besoin métier.

Normaliser

Bénéfices

- Cohérence : un modèle normalisé assurera la cohérence des données ;
- Requêtage : on garantit la possibilité de tout requêter si le langage supporté par la DB le permet ;
- Taille : la donnée est plus facilement distribuée dans le *cluster* et occupe moins d'espace.

Inconvénients

- Complexité : certaines requêtes peuvent être beaucoup plus complexes.

Pour résumer il semble sage de normaliser lorsque :

- On privilégie la cohérence;
- Certains *use cases* de dénormalisation pourraient faire grossir le document de manière infinie comme stocker une conversation entre deux individus, par exemple.

Le modèle évènementiel

Voici un cas de modélisation un peu moins traditionnel puisqu'il ne repose pas sur la modification d'un état existant. Ici on va stocker des évènements de création, modification ou suppression au lieu de mettre à jour un objet. On passe alors dans un modèle *append-only*. On ne fait qu'écrire de nouveaux objets représentant des évènements. On parle alors d'*Event Sourcing*. Le premier avantage est que vous ne perdez jamais d'information. Puisque vous connaissez chaque évènement, vous pouvez connaître l'état d'un objet à n'importe quel moment dans le temps. Ceci peut être particulièrement utile pour des logs d'audit, pour la traçabilité de la donnée.

Si on ne modifie pas d'état, alors il est facile de distribuer la donnée et donc d'*out-scaler* votre *cluster*. Si vous connaissez la taille et la fréquence de vos évènements, il est aussi très simple de prédire les besoins de scalabilité de votre cluster.

L'*event sourcing* permet aussi de rejouer les évènements et de visualiser plus simplement ce qu'une évolution de votre logique métier pourrait donner.

Prenons comme exemple une gestion de compte bancaire. Vous savez cet exemple qu'on prend à l'école pour vous expliquer l'*ACID*. Eh bien en pratique, c'est un très mauvais exemple.

En pratique un système bancaire n'utilise pas de transaction mais un journal de log, c'est à dire de l'*event sourcing*. On note les différentes opérations effectuées, puis on effectue une réconciliation.

Pouvez-vous imaginer le temps qu'il faudrait à un système massivement distribué comme le système bancaire mondial pour atteindre un *quorum* complet ? Cela semble très, très long. Surtout si on se place dans les années 80, bien avant les avancées technologiques que nous connaissons aujourd'hui.

LA DATA DANS LE CLOUD

La complexité sous-jacente à la gestion de la donnée a naturellement conduit les fournisseurs de solutions *cloud* à proposer des solutions intégrées, que l'on range depuis quelques temps dans le *serverless*.

Le marché

Dans cette catégorie, il est important de distinguer deux types d'offres : les bases de données managées, mais qui existent pour installation en local ou chez un autre hébergeur, et les bases de données qui existent exclusivement *as a service*, chez un fournisseur unique.

La pertinence de ces acteurs est évidente : ils savent gérer des infrastructures performantes, ont des process d'administration industrialisés et efficaces. Par ailleurs, héberger la donnée au plus près des capacités de traitement est plutôt souhaitable pour les questions de latence réseau dont nous parlions en début de dossier. Les consommateurs *IaaS* sont donc bien servis avec ces bases managées. Mais aussi les utilisateurs de nouveaux services cognitifs à base de *machine learning* et d'intelligence artificielle. Là encore, les données proximales offrent les meilleures performances.

Le premier cas est simple de mise en œuvre et de compréhension : il s'agit de fournir une base de données du marché (souvent *open source*, mais aussi sous licence payante) en intégrant un package de clustering *master-slave*, tout d'abord, puis de *monitoring*, *backups*, relance en cas d'incident et de mise à jour, chiffrement de la donnée *at-rest*. Une sorte de standard achetable en un click de la prestation d'un infogéreur traditionnel. Les offres sont variablement disponibles, en fonction du moteur de persistance de données et du niveau de qualité attendue. Ainsi *PostgreSQL*, *Redis*, *Memcached* ou *MySQL* sont des offres que l'on peut trouver facilement chez *Amazon Web Services*, *Google Cloud Platform*, *Microsoft Azure*, *Heroku* ou bien *Clever Cloud(16)* en France. Des offres de *Cassandra* ou de *Riak as a service* sont plus difficiles à trouver.

Dans le même cadre, on peut également trouver des *middleware* tel que *brokers* de messages (*RabbitMQ*, *HornetQ*, *Kafka*…) ou alors des moteurs de recherche (*Solr*, *Elasticsearch*…). Les offres se différencient souvent sur le niveau de garantie qu'elles apportent, ce qui oblige à se plonger dans les conditions générales de vente, ainsi que les ressources physiques associées. Certains fournisseurs appuyant leurs offres sur des machines virtuelles et d'autres sur des *containers* dans des OS mutualisés. Il est également intéressant de noter que ces offres peuvent être construites avec ou en opposition à l'éditeur de la solution proposée, permettant l'accès ou non aux fonctionnalités *Entreprise* et au support éditeur. Ces solutions sont toujours construites sur le principe *mono-tenant* de fournir une instance ou *cluster* par client, ne privilégiant pas le partage de ressources et garantissant un modèle d'accès dédié à la donnée. Il est bien entendu très simple de construire un plan de réversibilité ou de migration étant donné que le système est standard.

Restez vigilant malgré tout. Chez les *cloud providers*, disposer d'une instance de base de données *MySQL*, par exemple, ne veut pas nécessairement dire disposer de toutes les fonctionnalités natives du moteur. Le diable est dans les détails et utiliser une base de données *open* dans le *cloud*, c'est utiliser un moteur qu'on connait presque à 100%, mais parfois, certains tweaks ou certaines limitations existent.

Attention au lock-in applicatif…

L'autre typologie des offres est regroupée dans un nouvel ensemble : les bases de données *multi-tenant*, dont les éditeurs sont aussi les seuls exploitants. Ainsi la plus emblématique est probablement *Amazon S3(17)*, qui fournit une sorte de clé/valeur géant, conçu pour le stockage de fichiers (lourds ou non) et permettant un accès

(16) Disclaimer, il n'est pas impossible que les auteurs de ce dossier travaille pour Clever Cloud.
(17) Simple Server Storage

direct en *http* (concept communément appelé *object storage*). *S3* est construit comme une base unique, multi-régions, opérée par *AWS*. Si, dans le cas de *S3*, l'*API* du service est devenue un standard souvent réimplémenté par des clones plus ou moins performants en *open source* ou sous licence, cela reste l'exception de ce type d'offre. Ainsi *AuroraDB*, la base de données *MySQL* "distribuée" d'*AWS* utilise une *API* compatible avec l'*API* de *MySQL*, mais avec des limitations. DynamoDB, la base de données document d'*AWS* dispose d'une *API* "proche" de celle de *MongoDB*. *HMR* d'*AWS* est un *revamp* maison d'une distribution *Hadoop MapR*.

Les solutions telles que *DynamoDB* et *AuroraDB* (*AWS*), *Spanner*, *BigTable* ou *Firebase* (*GCP*), *CosmosDB* (*Ms Azure*) ou *Watson* (*IBM BlueMix*) sont des solutions qui ne permettent en aucun cas la migration vers d'autres possibilités d'hébergement sans migrer la donnée et réécrire sérieusement la couche d'accès à la donnée de l'application. Ceci créant une forme de *lock-in* dans l'IT, souvent mal compris par les preneurs de décision, et rendant le plan de réversibilité compliqué à évaluer. Par ailleurs, il existe une équation économique relativement complexe à estimer en début de projet. Les modèles tarifaires sont multiples, mais peuvent reposer sur des éléments difficiles à quantifier comme le volume de données remonté à chaque requête, ou bien la durée de traitement de la requête… Enfin, il faut aussi tenir compte du fait qu'une base de données disponible uniquement sur le *cloud* implique également… que chaque développeur doit disposer d'une instance dans le *cloud*, se conformer à une stratégie d'entreprise, là où Docker lui avait redonné de la latitude dans la gestion de son environnement de dev.

En revanche, ces bases de données peuvent s'avérer extrêmement pertinentes dans certains use cases particuliers, notamment car leur principe *multi-tenant* leur donne des capacités de scalabilité très véloce dans une vraie simplicité. Ainsi, *BigTable* est capable de restituer le résultat de requêtes complexes sur des To de données en quelques dixièmes de secondes.

Dans les offres des *cloud providers*, on retrouve des SGBD-R, avec *RDS* (*AWS*), *Cloud SQL* (*GCP*), *SQL Databases* (*Azure*) ; on retrouve des bases clé-valeur *Elasticache* (*AWS*), *Table storage* (*Azure*), des bases orientées colonnes *Redshift* (*AWS*), *BigTable* (*GCP*), *CosmosDB* (*Azure*), des bases orientées graphe *GremlinsDB* (*Azure*). On trouve aussi ces tentatives d'outrepasser le *CAP theorem* en proposant des SGBD-R optimisées sur des infrastructures réseau ultra-performantes AuroraDB (AWS), Spanner (GCP), CosmosDB (Azure). Au-delà des effets d'annonce *marketing*, ces systèmes très performants repoussent les limites perceptibles du *CAP theorem*, mais sans en lever les contraintes. Les coûts étant par ailleurs assez élevés, il conviendra de faire des tests poussés au moment du choix.

… et au lock-in physique

On regroupe souvent l'ensemble de ces offres de base de données dans le *cloud* sous le terme de *DBaaS*. Vos applications utiliseront alors une base de données dans le *cloud*. Suivant votre cas d'utilisation, vous devrez faire attention à la latence que cela induit. Imaginons que vos applications soient déployées dans un *datacenter* à Marseille, et qu'elles utilisent une *DBaaS* qui se trouve physiquement en Irlande ou même aux Etats-Unis. Outre les problèmes légaux que cela peut poser (là c'est un autre dossier sur la gouvernance des données, la RGPD vous connaissez ?), vous ferez face à un problème de latence. S'il y a bien une partie du temps de la requête que vous ne pouvez pas réduire, c'est la durée que mettent vos paquets réseaux à traverser des câbles. Donc quand vous choisissez une *DBaaS*, faites en sorte de pouvoir déployer vos applications à côté.

Et pendant que nous parlons de détails physiques, abordons une autre notion, la Gravité de la donnée. La donnée à un poids.

Avec l'*Infra as Code* et l'automatisation des déploiements, migrer une application d'un *cloud provider* vers un autre est relativement trivial : quelques jours de travail maximum.

Imaginez-vous devoir migrer une ou plusieurs bases d'un *cloud provider* à l'autre sans interruption de service ? Vous devrez bouger toutes vos données. Tout ceci prend un certain temps. On retombe sur les limites physiques de notre ami le câble réseau. Lorsque vous choisissez une solution de *DBaaS* ou un fournisseur cloud pour héberger vos données, vous faites un choix structurant. Pour se battre contre cette problématique, certaines bases de données ont choisi une approche applicative en permettant de faire de la réplication managée entre *cluster* ou entre nœuds d'un même *cluster*. Regardez ce que propose *Couchbase*, *RedisLabs* ou encore *Cassandra*.

LE POINT BIG DATA

Si jusqu'ici nous avons essentiellement parlé de base de données, nous n'avons pas trop abordé le sujet *Big Data*. Mais à quel moment parlons-nous de *Big Data* ?

"Je fais du Big Data, regarde donc cette courbe magnifique réalisée avec Tableau à partir de cette feuille Excel qui tient sur une clé USB."

Pour avoir déjà entendu ce genre de choses en France, je sais qu'il y a un biais cognitif sur le terme *Big Data*. La plupart de mes interlocuteurs me parlent de *reporting*, de *BI*, d'*analytics*, de *datasets* d'entraînement de traitement de *machine learning* ou d'IA, c'est-à-dire des métiers associés à la *Big Data*.

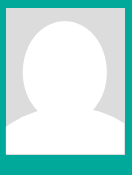
Alors que la *Big Data* (c'est marqué dessus) est avant tout un sujet technologique : comment stocker et traiter une grande quantité d'informations, là où des enjeux d'architecture forts se posent.

Avec l'avènement des solutions *NoSQL* et des *DBaaS*, la frontière de la *Big Data* s'est vue repoussée… Certaines technologies comme *Cassandra*, *Redshift* (*AWS*), *BigTable* (*GCP*) flirtent avec le *Big Data*, au *petaoctet*. Au-delà, les maîtres incontestés restent l'écosystème *Hadoop* pour le stockage et le traitement de données dénormalisées et *AWS S3* dans le domaine de l'*object storage*.

TOUT CELA EST BIEN GENTIL MAIS JE CHOISIS COMMENT ?

Excellente question qui en amène beaucoup d'autres. Voici une tentative de grille non exhaustive pour effectuer un choix de base de données. Gardez à l'esprit que vous pouvez choisir plusieurs outils pour gérer vos données. Essayez donc de décomposer au maximum vos choix suivant ce que vous devez requêter. Ensuite, demandez-vous :

- Quel sera le ratio lecture/écriture ?
- La cohérence de la donnée est-elle obligatoire ?
- Ai-je besoin de transactions ?
- Dois-je privilégier la rapidité à la cohérence ?
- Si la rapidité est importante ;
  - La donnée ou les index peuvent-ils être en RAM ?
  - Comment peuvent être distribuées les données géographiquement ?


Sylvain SAUREL
Ingénieur d'Etudes Java / Android
sylvain.saurel@gmail.com
https://www.ssaurel.com/blog

Créez votre propre Blockchain en Java

La folie Bitcoin qui s'est emparée du monde à la fin de l'année 2017 aura permis de mettre en lumière les crypto-monnaies auprès du grand public. Au cœur du Bitcoin et des crypto-monnaies, on trouve la technologie Blockchain qui est une véritable révolution technologique, comparable à ce que fut Internet en son temps pour beaucoup. Dans cet article, nous vous proposons de créer votre propre Blockchain en Java afin de découvrir les rouages internes de cette technologie révolutionnaire.

La Blockchain, que l'on peut traduire par chaîne de blocs dans la langue de Molière, est une technologie de stockage et de transmission d'informations sans aucun organe de contrôle. Techniquement parlant, la Blockchain peut être vue comme une base de données distribuée au sein de laquelle les informations transmises par les utilisateurs (les transactions dans le cas du Bitcoin) sont vérifiées et groupées à intervalles de temps réguliers en blocs. 1

Ces blocs sont liés et sécurisés grâce à l'utilisation de la cryptographie et forment ainsi une chaîne de blocs : la fameuse Blockchain. Celle-ci pouvant être vue comme un registre distribué et sécurisé de toutes les transactions effectuées depuis le démarrage du système réparti.

Au même titre qu'Internet en son temps, la Blockchain constitue une véritable révolution qui permet de faire ce qu'Internet n'aura finalement jamais été capable de réaliser jusqu'alors. En effet, alors qu'Internet autorise l'échange d'informations de manière décentralisée, nous n'avons pu nous passer de plateformes faisant office de tiers de confiance lorsqu'il s'est agi d'envoyer de l'argent par Internet. La Blockchain résout ce problème en rendant caduc le besoin d'un intermédiaire de confiance. Le passage par un intermédiaire comme Paypal lors d'une transaction entre deux personnes devient ainsi inutile. Avec la Blockchain, ce sont les ordinateurs du réseau qui vont vérifier si les deux personnes sont fiables afin de valider leur transaction. La chaîne de blocs certifiant tout le processus en s'appuyant sur la contribution des ordinateurs du réseau.

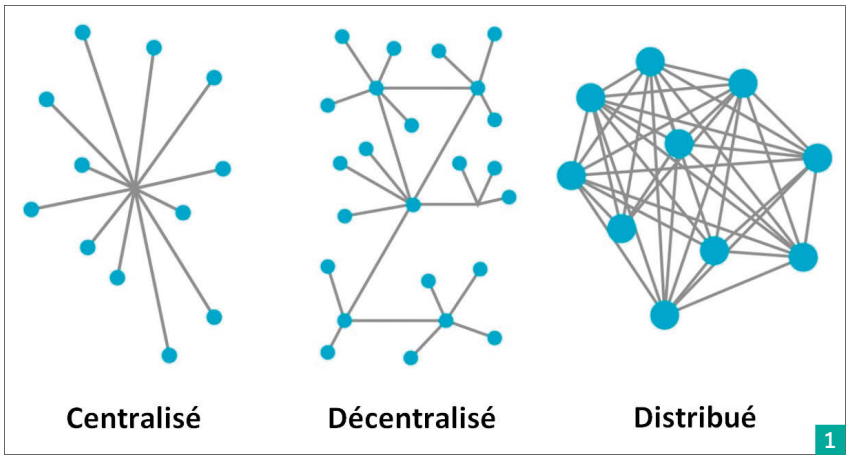
Dans la suite de cet article, nous allons implémenter notre propre Blockchain en local afin de mettre en lumière les rouages internes de cette technologie. Pour ce faire, nous nous appuierons sur le langage Java.

Modélisation et création des Blocs

La Blockchain étant une chaîne de blocs, nous commençons par modéliser et créer les blocs qui vont la composer. De manière basique, un bloc contient les informations suivantes :

- Un index ;
- Un timestamp représentant sa date de création ;
- Le Hash du précédent bloc ;
- Des données stockées au sein du bloc. Dans le cadre du Bitcoin et des autres crypto monnaies, il s'agira donc de transactions ;
- Le Hash du bloc courant permettant de garantir l'intégrité de son contenu. 2

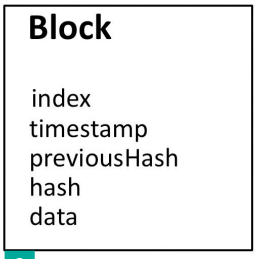
Ceci nous donne donc le code suivant pour les propriétés de notre classe Block :



1

La Blockchain est une BD distribuée

```
public class Block {

 private int index;
 private long timestamp;
 private String hash;
 private String previousHash;
 private String data;
 private int nonce;

 // ...
}
```



2 *Modélisation d'un Bloc*

Fonction de hachage cryptographique SHA-256

Le Hash des blocs de notre Blockchain est calculé en s'appuyant sur un algorithme cryptographique de la famille SHA-2 : le SHA-256. Par chance, Java propose en standard une implémentation de cet algorithme de hachage. Nous n'aurons ainsi pas besoin de le coder nous-même. Les algorithmes de cryptographie implémentés au sein du JDK peuvent être récupérés via la classe MessageDigest et sa méthode statique getInstance() à laquelle il suffit de passer en entrée le nom de l'algorithme souhaité.

Il reste ensuite à passer en entrée de la méthode digest() de l'instance de MessageDigest récupérée une représentation textuelle du contenu du bloc à hacher pour obtenir en sortie le résultat de son hachage via l'algorithme SHA-256 sous la forme d'un tableau de bytes. Enfin, on transforme ce tableau de bytes en chaîne de caractères avant de renvoyer celle-ci en sortie de fonction :

```
public static String calculateHash(Block block) {
 if (block != null) {
```

```
  MessageDigest digest = null;

  try {
   digest = MessageDigest.getInstance("SHA-256");
  } catch (NoSuchAlgorithmException e) {
   return null;
  }

  String txt = block.str();
  final byte bytes[] = digest.digest(txt.getBytes());
  final StringBuilder builder = new StringBuilder();

  for (final byte b : bytes) {
   String hex = Integer.toHexString(0xff & b);

   if (hex.length() == 1) {
    builder.append('0');
   }

   builder.append(hex);
  }

  return builder.toString();
 }

 return null;
}
```

Minage d'un Bloc

Le bloc que nous venons de créer est quasiment fonctionnel. Il ne nous reste plus qu'à lui ajouter une méthode permettant de réaliser son minage. Le processus de minage va consister à résoudre une énigme posée par la fameuse "Proof of Work", connue également sous le nom de preuve de travail en Français. L'énigme mathématique consiste à trouver un hachage pour le bloc commençant par un nombre de zéros donnés. Bien entendu, la difficulté de résolution de l'énigme est proportionnelle au nombre de zéros initiaux souhaités pour le hachage du bloc. Il est important de comprendre également qu'une difficulté de résolution accrue va augmenter le besoin en puissance de calculs d'ordinateurs à mobiliser. Ce point étant au cœur d'une des problématiques de la Blockchain Bitcoin notamment puisque plus le nombre de jetons Bitcoin en circulation augmente, plus la difficulté pour en miner des nouveaux va augmenter. La puissance de calculs à mobiliser s'envole, ce qui entraîne une consommation électrique plus importante et donc un minage toujours plus coûteux.
Tout ceci nous donne l'implémentation suivante pour notre méthode de minage au sein de la class Block :

```
public void mineBlock(int difficulty) {
 nonce = 0;

 while (!getHash().substring(0, difficulty).equals(Utils.zeros(difficulty))) {
  nonce++;
  hash = Block.calculateHash(this);
 }
}
```

Précisons que la propriété nonce de l'objet Block sert à stocker le nombre d'essais réalisés avant la résolution de la preuve de travail durant le minage. En outre, la méthode statique zeros de la classe Utils est utilisée ici pour retourner une chaîne de caractères contenant le nombre de zéros passé en paramètre en entrée :

```
public class Utils {
 public static String zeros(int length) {
  StringBuilder builder = new StringBuilder();

  for (int i = 0; i < length; i++) {
   builder.append("0");
  }

  return builder.toString();
 }
}
```

Code complet de la classe Block

Notre objet Block, modélisant un bloc de notre Blockchain, sera ainsi implémenté de la sorte :

```
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Date;

public class Block {

 private int index;
 private long timestamp;
 private String hash;
 private String previousHash;
 private String data;
 private int nonce;

 public Block(int index, long timestamp, String previousHash, String data) {
  this.index = index;
  this.timestamp = timestamp;
  this.previousHash = previousHash;
  this.data = data;
  nonce = 0;
  hash = Block.calculateHash(this);
 }

 public int getIndex() {
  return index;
 }

 public long getTimestamp() {
  return timestamp;
 }

 public String getHash() {
  return hash;
 }

 public String getPreviousHash() {
```

```
  return previousHash;
 }

 public String getData() {
  return data;
 }

 public String str() {
  return index + timestamp + previousHash + data + nonce;
 }

 public String toString() {
  StringBuilder builder = new StringBuilder();
  builder.append("Block #").append(index).append(" [previousHash : ").append(previous
Hash).append(", ").
  append("timestamp : ").append(new Date(timestamp)).append(", ").append("data : ").
append(data).append(", ").
  append("hash : ").append(hash).append("]");
  return builder.toString();
 }

 public static String calculateHash(Block block) {
  if (block != null) {
   MessageDigest digest = null;

   try {
    digest = MessageDigest.getInstance("SHA-256");
   } catch (NoSuchAlgorithmException e) {
    return null;
   }

   String txt = block.str();
   final byte bytes[] = digest.digest(txt.getBytes());
   final StringBuilder builder = new StringBuilder();

   for (final byte b : bytes) {
    String hex = Integer.toHexString(0xff & b);

    if (hex.length() == 1) {
     builder.append('0');
    }

    builder.append(hex);
   }

   return builder.toString();
  }

  return null;
 }

 public void mineBlock(int difficulty) {
  nonce = 0;

  while (!getHash().substring(0,  difficulty).equals(Utils.zeros(difficulty))) {
   nonce++;
   hash = Block.calculateHash(this);
  }
 }

}
```

Modélisation de la Blockchain

Comme dit précédemment, la Blockchain est une base de données distribuée garantissant l'intégrité des données la composant. Nous allons donc pouvoir passer à l'implémentation de l'objet Blockchain. Ce dernier va tout d'abord se voir doter d'une propriété permettant de définir le niveau de difficulté que l'on souhaite pour le minage des blocs durant la résolution de la preuve de travail. Ensuite, la Blockchain va stocker ces blocs au sein d'un objet List ce qui nous donne le début de déclaration suivant pour l'objet Blockchain :

```
public class Blockchain {
 private int difficulty;
 private List<Block> blocks;
}
```

Création et Minage de nouveaux Blocs

Une des missions de notre Blockchain est de permettre la création de nouveaux blocs ainsi que leur minage en tenant compte de la difficulté associée. Pour ce faire, nous ajoutons une méthode newBlock() permettant de générer un nouveau bloc qui sera lié au dernier bloc connu de la Blockchain. Le minage du bloc se faisant quant à lui au moment de l'ajout du bloc à la Blockchain dans une méthode dédiée addBlock() :

```
public Block latestBlock() {
 return blocks.get(blocks.size() - 1);
}

public Block newBlock(String data) {
 Block latestBlock = latestBlock();
 return new Block(latestBlock.getIndex() + 1, System.currentTimeMillis(),
   latestBlock.getHash(), data);
}

public void addBlock(Block b) {
 if (b != null) {
  b.mineBlock(difficulty);
  blocks.add(b);
 }
}
```

Vérification de la validité de la Blockchain

La Blockchain doit également garantir l'intégrité des données de ses blocs afin de rester valide. Dans ce but, nous allons ajouter différentes méthodes au sein de notre Blockchain pour vérifier que :

- Le premier bloc est valide ;
- Un nouveau bloc est valide avec le bloc précédent de la Blockchain ;
- La Blockchain est valide.

Avant d'implémenter ces méthodes au sein de notre objet

Blockchain, il convient de définir les critères de validité que nous allons considérer. Le premier bloc sera considéré comme valide si son index est égal à 0, qu'il ne possède pas de previousHash renseigné et enfin si son Hash est correct. Un nouveau bloc est valide en regard du bloc précédent de la Blockchain si son index est incrémenté de 1 par rapport à l'index du bloc précédent, son champ previousHash est renseigné avec le Hash du bloc précédent et enfin si son Hash est lui-même cohérent par rapport aux données qu'il contient. Enfin, la validité de la Blockchain ainsi que l'intégrité des données qu'elle contient peuvent être garanties lorsque le premier bloc est valide et que chaque bloc qui la compose est valide en regard du bloc qui le précède. Tout ceci nous donne le code suivant :

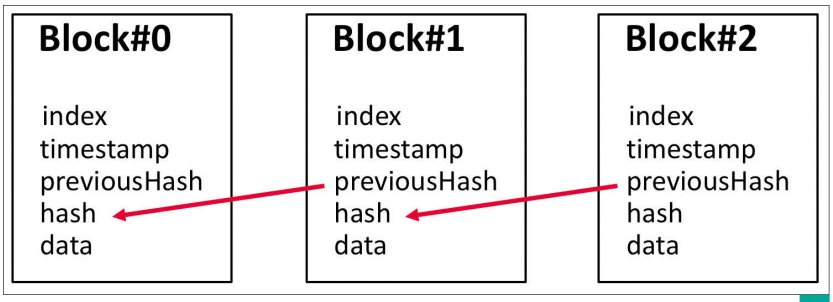
```
public boolean isFirstBlockValid() {
  Block firstBlock = blocks.get(0);

  if (firstBlock.getIndex() != 0) {
    return false;
  }

  if (firstBlock.getPreviousHash() != null) {
    return false;
  }

  if (firstBlock.getHash() == null ||
      !Block.calculateHash(firstBlock).equals(firstBlock.getHash())) {
    return false;
  }

  return true;
}

public boolean isValidNewBlock(Block newBlock, Block previousBlock) {
  if (newBlock != null  &&  previousBlock != null) {
    if (previousBlock.getIndex() + 1 != newBlock.getIndex()) {
      return false;
    }

    if (newBlock.getPreviousHash() == null  ||
        !newBlock.getPreviousHash().equals(previousBlock.getHash())) {
      return false;
    }

    if (newBlock.getHash() == null  ||
        !Block.calculateHash(newBlock).equals(newBlock.getHash())) {
      return false;
    }

    return true;
  }

  return false;
}

public boolean isBlockChainValid() {
  if (!isFirstBlockValid()) {
    return false;
  }

  for (int i = 1; i < blocks.size(); i++) {
    Block currentBlock = blocks.get(i);
    Block previousBlock = blocks.get(i - 1);

    if (!isValidNewBlock(currentBlock, previousBlock)) {
      return false;
    }
  }

  return true;
}
```

Code complet de la classe Blockchain

Notre objet Blockchain est pratiquement terminé. Il ne reste plus qu'à surcharger sa méthode toString() afin de pouvoir renvoyer une représentation textuelle facilitant la visualisation de son contenu. Tout ceci nous donne le code suivant :

```
import java.util.ArrayList;
import java.util.List;

public class Blockchain {

  private int difficulty;
  private List<Block> blocks;

  public Blockchain(int difficulty) {
    this.difficulty = difficulty;
    blocks = new ArrayList<>();
    // create the first block
    Block b = new Block(0, System.currentTimeMillis(), null, "First Block");
    b.mineBlock(difficulty);
    blocks.add(b);
  }

  public int getDifficulty() {
    return difficulty;
  }

  public Block latestBlock() {
    return blocks.get(blocks.size() - 1);
  }

  public Block newBlock(String data) {
    Block latestBlock = latestBlock();
    return new Block(latestBlock.getIndex() + 1, System.currentTimeMillis(),
        latestBlock.getHash(), data);
  }

  public void addBlock(Block b) {
    if (b != null) {
      b.mineBlock(difficulty);
      blocks.add(b);
    }
```

```
}

public boolean isFirstBlockValid() {
 Block firstBlock = blocks.get(0);

 if (firstBlock.getIndex() != 0) {
  return false;
 }

 if (firstBlock.getPreviousHash() != null) {
  return false;
 }

 if (firstBlock.getHash() == null ||
     !Block.calculateHash(firstBlock).equals(firstBlock.getHash())) {
  return false;
 }

 return true;
}

public boolean isValidNewBlock(Block newBlock, Block previousBlock) {
 if (newBlock != null  &&  previousBlock != null) {
  if (previousBlock.getIndex() + 1 != newBlock.getIndex()) {
   return false;
  }

  if (newBlock.getPreviousHash() == null  ||
   !newBlock.getPreviousHash().equals(previousBlock.getHash())) {
   return false;
  }

  if (newBlock.getHash() == null  ||
   !Block.calculateHash(newBlock).equals(newBlock.getHash())) {
   return false;
  }

  return true;
 }

 return false;
}

public boolean isBlockChainValid() {
 if (!isFirstBlockValid()) {
  return false;
 }

 for (int i = 1; i < blocks.size(); i++) {
  Block currentBlock = blocks.get(i);
  Block previousBlock = blocks.get(i - 1);

  if (!isValidNewBlock(currentBlock, previousBlock)) {
   return false;
  }
 }
```



Vue simplifiée d'une Blockchain 3

```
  return true;
 }

 public String toString() {
  StringBuilder builder = new StringBuilder();

  for (Block block : blocks) {
   builder.append(block).append("\n");
  }

  return builder.toString();
 }
}
```

Notre Blockchain entre en action

La dernière étape va consister à mettre notre Blockchain en action. Pour cela, nous allons instancier un objet Blockchain en définissant en entrée une difficulté de 4 pour le minage des blocs. Ensuite, nous ajouterons trois blocs avant de vérifier la validité de la Blockchain ainsi créée et d'afficher les données qu'elle contient à l'écran. 3

Le code suivant permet donc d'effectuer ces manipulations sur notre Blockchain :

```
public class Main {

 public static void main(String[] args) {
  Blockchain blockchain = new Blockchain(4);
  blockchain.addBlock(blockchain.newBlock("SSaurel's Blog"));
  blockchain.addBlock(blockchain.newBlock("Sylvain Saurel"));
  blockchain.addBlock(blockchain.newBlock("https://www.ssaurel.com/blog"));

  System.out.println("Blockchain valid ? " + blockchain.isBlockChainValid());
  System.out.println(blockchain);
 }

}
```

L'exécution de ce code va nous donner la sortie console présentée à la figure 4 sur laquelle on peut constater visuellement la validité de notre Blockchain. 4

Corruption de notre Blockchain

Afin de vérifier le bon fonctionnement de notre méthode de validation de la Blockchain, il peut être intéressant de tenter de la corrompre en lui intégrant un bloc corrompu. Une fois cet ajout

```
Console  Javadoc  Search  Progress  Error Log  Declaration  LogCat
<terminated> Main (24) [Java Application] C:\Program Files\Java\jre1.6.0_18\bin\javaw.exe (8 févr. 2018 13:48:36)
Blockchain valide ? true
Block #0
 [previousHash : null,
timestamp : Thu Feb 08 13:48:36 CET 2018,
data : First Block,
hash : 0000049bf9589d61a8e8fd04cf4c3210290a04748ab1f7900ed6bf6651527888]

Block #1
 [previousHash : 0000049bf9589d61a8e8fd04cf4c3210290a04748ab1f7900ed6bf6651527888,
timestamp : Thu Feb 08 13:48:36 CET 2018,
data : SSaurel's Blog,
hash : 0000a50f0b3d5ae078c5932e2ebb64cfbd86ed63fe7a3dfa34f428d238969 0f2]

Block #2
 [previousHash : 0000a50f0b3d5ae078c5932e2ebb64cfbd86ed63fe7a3dfa34f428d238969 0f2,
timestamp : Thu Feb 08 13:48:37 CET 2018,
data : Sylvain Saurel,
hash : 0000f70de7b6c8c45cf4021fa7633995e970f10b3da07ae26999149f35bb5057]

Block #3
 [previousHash : 0000f70de7b6c8c45cf4021fa7633995e970f10b3da07ae26999149f35bb5057,
timestamp : Thu Feb 08 13:48:37 CET 2018,
data : https://www.ssaurel.com/blog,
hash : 00001f4bd2f38b7efc2757f1a868f460016e58f4250861f23ee2b91f2bd18341]
```

4 *Exécution de notre Blockchain*

```
Console  Javadoc  Search  Progress  Error Log  Declaration  LogCat
<terminated> Main (24) [Java Application] C:\Program Files\Java\jre1.6.0_18\bin\javaw.exe (8 févr. 2018 14:00:44)
Blockchain valide ? false
Block #0
 [previousHash : null,
timestamp : Thu Feb 08 14:00:44 CET 2018,
data : First Block,
hash : 0000305298b7fdad0b09229a96e6b355019ffc57bddfa82a857a8dd1c6ce79db]

Block #1
 [previousHash : 0000305298b7fdad0b09229a96e6b355019ffc57bddfa82a857a8dd1c6ce79db,
timestamp : Thu Feb 08 14:00:44 CET 2018,
data : SSaurel's Blog,
hash : 0000a2153a3315afd077da38ddd6f857798054602546b8b7fc36823fb007a71e]

Block #2
 [previousHash : 0000a2153a3315afd077da38ddd6f857798054602546b8b7fc36823fb007a71e,
timestamp : Thu Feb 08 14:00:45 CET 2018,
data : Sylvain Saurel,
hash : 00007b18722eb2dd088cf200ca12b6b5f10e0e96a477e765f16d59161b27f32a]

Block #3
 [previousHash : 00007b18722eb2dd088cf200ca12b6b5f10e0e96a477e765f16d59161b27f32a,
timestamp : Thu Feb 08 14:00:45 CET 2018,
data : https://www.ssaurel.com/blog,
hash : 00004ddd275a06da8ad72e582e9270c54362a4ecd090bd3dae0843567b5618fb]

Block #15
 [previousHash : aaaabbb,
timestamp : Thu Feb 08 14:00:46 CET 2018,
data : Block invalide,
hash : 0000e8b4bf6b40ca06f6c4544ffdd875dbafb4e9daa17b9fe543c34cba32ec37]
```

Bloc corrompu

5 *Détection du bloc corrompu*

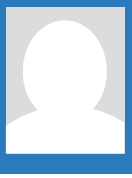
réalisé, nous pourrons vérifier que notre méthode détecte bien que la Blockchain n'est désormais plus valide :

```
public class Main {

 public static void main(String[] args) {
  Blockchain blockchain = new Blockchain(4);
  blockchain.addBlock(blockchain.newBlock("SSaurel's Blog"));
  blockchain.addBlock(blockchain.newBlock("Sylvain Saurel"));
  blockchain.addBlock(blockchain.newBlock("https://www.ssaurel.com/blog"));

  System.out.println("Blockchain valide ? " + blockchain.isBlockChainValid());
  System.out.println(blockchain);

  // Ajout d'un bloc corrompu à notre Blockchain
  blockchain.addBlock(new Block(15, System.currentTimeMillis(), "aaaabbb", "Block invalide"));

  System.out.println("Blockchain valide ? " + blockchain.isBlockChainValid());
 }
}
```

L'exécution de ce code va produire la sortie console présentée à la figure 5. On peut remarquer que notre Blockchain détecte bien un problème de validité avec le bloc corrompu que nous avons ajouté.

Conclusion

Notre implémentation d'une Blockchain locale en Java est pleinement fonctionnelle comme nous avons pu le constater. Celle-ci nous aura permis de mettre en évidence le fonctionnement interne d'une Blockchain et notamment la partie liée au minage de blocs qui se révèle d'autant plus rapide que la puissance de calculs des ordinateurs la réalisant est importante. Pour aller plus loin, il nous faudrait maintenant implémenter les fonctionnalités permettant la mise en réseau pair-à-pair (P2P) de notre Blockchain. Cette mise en réseau pourra être proposée dans un futur article. •



Alexis CONIA
Développeur et entrepreneur
MVP Office Servers and Services
www.xpanduc.com

Microsoft Teams pour les développeurs

Annoncé il y a un peu plus d'un an, Microsoft Teams est le nouvel outil de collaboration de la suite Office 365. Il redéfinit la façon de travailler et de collaborer avec ses collègues via différentes équipes et canaux. Basé sur un modèle extensible, les développeurs y trouvent également leur intérêt via les différentes options pour intégrer leur application.

Vu comme le concurrent direct de Slack lors de sa sortie, Microsoft Teams a fait un petit bout de chemin depuis et s'éloigne de plus en plus de cette autre plateforme. En effet, Teams profite directement de toute la variété et la puissance de tous les composants présents dans Office 365 comme la suite Office ou bien encore des services comme Planner et Skype.

Rapide Présentation

Les collaborateurs sont regroupés en équipe et chaque sujet est découpé dans un canal. Chacun contient un fil de discussion et différents onglets pour présenter des fichiers ou des applications.

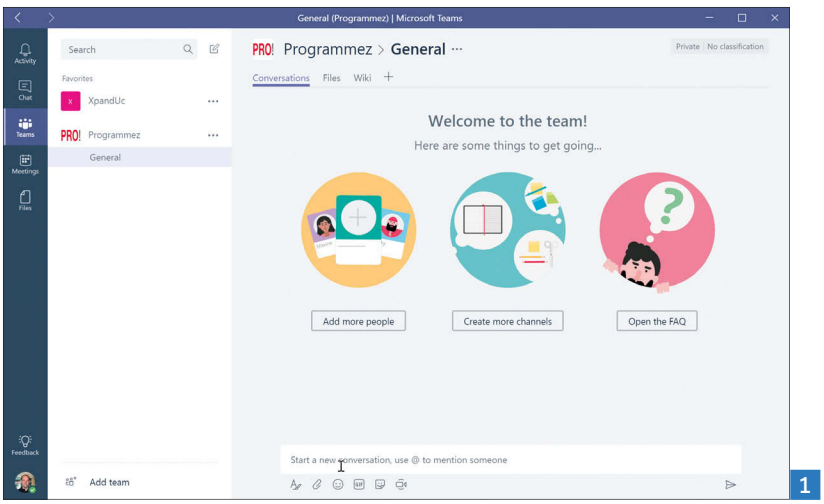
L'interface est composée de plusieurs sections : Activité, Conversations, Equipe, Réunions, Fichiers 1

Il est présenté comme le hub pour la collaboration et c'est bien le cas : il utilise Exchange pour créer un groupe et gérer la sécurité, un site d'équipe Sharepoint pour stocker les données ou bien encore le moteur de Skype pour communiquer en temps réel avec les autres membres de l'équipe. Point important, Microsoft a d'ailleurs annoncé lors de l'Ignite en septembre 2017 que Teams remplacera à terme le service Skype for Business Online (https://blogs.office.com/en-us/2017/09/25/a-new-vision-for-intelligent-communications-in-office-365/?eu=true)

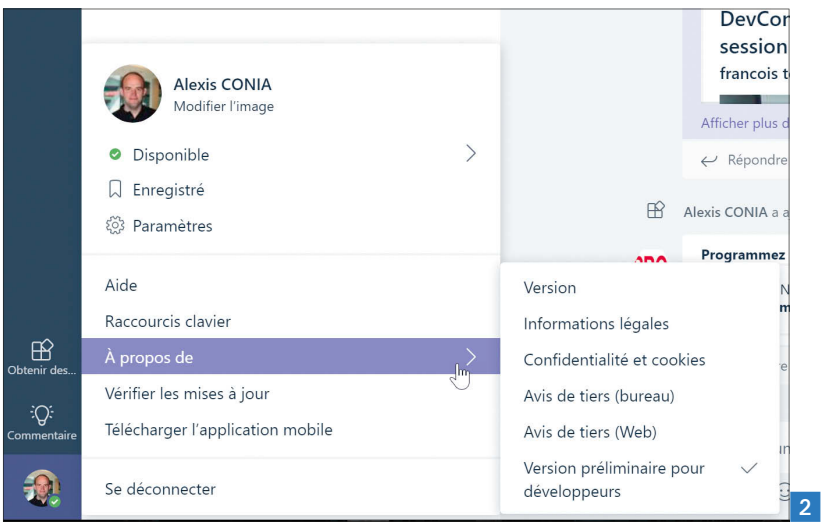
Capacité de développement

Plusieurs options sont disponibles :

- Onglet,
- Bot,
- Connecteur,
- Extension des messages,
- Microsoft Graph.



1



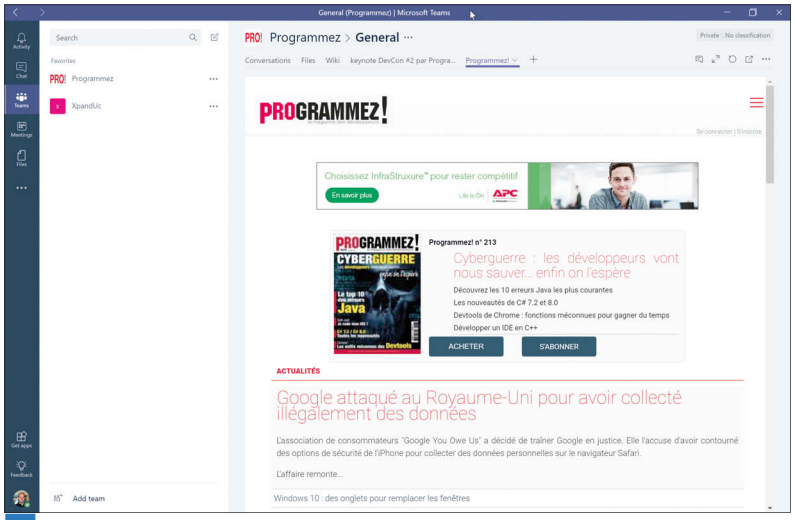
2

Une partie est encore en preview. Pour pouvoir utiliser toutes les fonctionnalités, le client peut être passé en mode développeur. Voici la démarche : cliquez sur votre photo en bas à gauche et cliquez sur « À propos de » puis « Version préliminaire pour développeur ». Votre client va redémarrer et sera en mode développeur. 2

Entrons un peu plus dans le détail de chaque option.

Onglet

Il est possible d'ajouter deux types d'onglets : le premier, de type « statique », permet d'épingler un site dans un canal (ex : le site de programmez !). Le deuxiè-



3 *Par exemple, le site de Programmez! est ici épinglé.*

me, plus flexible, ajoute une partie configuration et offre la possibilité au site de connaître le contexte dans lequel il est affiché (Nom de l'équipe, du canal, langage, thème, etc. …). Il existe un SDK JavaScript pour récupérer toutes ces informations sur le site, mais aussi pour gérer l'authentification dans Teams si nécessaire. 3

Bot

Comme la plupart des solutions de communication Microsoft, Teams peut dialoguer avec un bot. Il est intégré directement dans le bot Framework. Cet outil a déjà été présenté à plusieurs reprises, nous éviterons donc de rentrer dans le détail de l'implémentation. Un utilisateur peut dialoguer avec le bot depuis la partie conversation de Teams ou via un canal. Il peut également envoyer un message pour vous avertir ou vous laisser une notification via la section Activité. Comme pour un onglet, une api spécifique (pour le bot builder) est disponible pour que le bot obtienne le contexte dans lequel il est appelé : informations sur le profil, les membres d'une équipe ou bien encore la liste des canaux.

Connecteur

Un connecteur est une adresse de webhook pour un canal. 4

Si nous prenons l'exemple d'une application Programmez, celle-ci pourrait poster à chaque nouveau numéro, un message vers tous les canaux qui ont ajouté le connecteur. Le design du message est personnalisable. Il est basé sur les cartes présentes également dans le bot framework. Il existe plusieurs types de cartes pour ajouter du texte, des liens, des images et des boutons correspondant à des actions prédéfinies (ouvrir une URL, envoyer un message).

Extension des messages

La zone de composition d'un message peut être enrichie. Ce message est une activité spécifique du bot framework et renvoie une carte comme dans la section précédente. 5 6

Vous pouvez formater le message comme vous le souhaitez et y inclure l'ensemble des informations et des actions nécessaires.

Microsoft Graph

Teams est aussi disponible à travers le graph. Il s'agit d'une bêta pour le moment, mais il expose des informations utiles pour gérer l'outil à partir d'une autre application ou d'un script.

L'API Rest permet de manipuler les équipes, les canaux et de créer des ressources. Vous pouvez tester directement les requêtes sur le graph explorer (https://developer.microsoft.com/en-us/graph/graph-explorer).

Fichier Manifest et Store

Toutes les informations et la description des différentes fonctionnalités de l'application sont décrites dans un fichier JSON. Le développeur package ensuite son application pour la soumettre sur le store Office. Si elle est déployée dans un environnement hors du store, elle peut aussi être en mode "sideloader" et mise à disposition de l'équipe sans passer par le Store.

4

5

```
public async Task<HttpResponseMessage> Post([FromBody]Activity
activity)
{
    if (activity.Type == ActivityTypes.Invoke)
    {
        if (activity.IsComposeExtensionQuery())
        {
            var query = activity.GetComposeExtensionQueryData();
        }
    } else {
      // Failure case catch-all.
      var response =
Request.CreateResponse(HttpStatusCode.BadRequest);
      response.Content = new StringContent("Invalid request! This
      API supports only messaging extension requests. Check your
      query and try again");
      return response;
    }
}
```

6 *Exemple d'un message riche pour poster une vidéo Youtube dans un canal.*

Conclusion

Comme évoqué dans l'introduction, Teams va probablement devenir l'outil indispensable de collaboration dans l'environnement Microsoft, il serait alors dommage de passer à côté de ce « canal » pour votre application. Vous pouvez retrouver un exemple complet d'une application « To-Do » sur le github Office Dev et démarrer rapidement le développement sous Microsoft Teams ! •

Liens :

• Doc Développeur Teams : https://docs.microsoft.com/en-us/microsoftteams/platform/overview

• Exemple d'application To-Do: https://github.com/OfficeDev/microsoft-teams-sample-todo

David Carteau
carteau.david@gmail.com

L'apprentissage incrémental à population Partie 2

L'apprentissage incrémental à population, en anglais « Population Based Incremental Learning » (PBIL), a été proposé en 1994 par Shumeet Baluja. Inspirée des algorithmes génétiques, cette méthode est présentée par son auteur comme étant la à fois plus simple à implémenter que ces derniers, tout en donnant des résultats plus rapides et plus précis.

Création d'une population

La création d'une nouvelle population ne pose aucune difficulté. Elle prend en paramètre deux arguments : le nombre d'individus devant composer la population, et la distribution de probabilité qui sera utilisée pour la création des individus. Afin de mettre à jour le vecteur de probabilité, il est nécessaire d'évaluer chaque individu de la population puis d'identifier au sein de cette population l'individu le mieux adapté ainsi que celui le moins adapté :

```
public class Population {

    private static int iteration = 0;
    private Individual[] individuals;

    private Individual winner = null;
    private Individual looser = null;

    public Population(int size, Vector vector) {
        iteration++;
        individuals = new Individual[size];

        for (int i = 0; i < size; i++) {
            individuals[i] = new Individual(vector, iteration + "-" + (i + 1));
        }
    }

    public void evaluate() {
        int winnerFitness = Integer.MIN_VALUE;
        int looserFitness = Integer.MAX_VALUE;

        for (int i = 0; i < individuals.length; i++) {
            individuals[i].evaluate();

            int fitness = individuals[i].getFitness();
            System.out.println("Fitness of " + individuals[i].getName() + " = " + fitness);

            if (fitness > winnerFitness) {
                winner = individuals[i];
                winnerFitness = fitness;
            }
            if (fitness < looserFitness) {
                looser = individuals[i];
                looserFitness = fitness;
            }
        }
    }

    public Individual getWinner() {
        return winner;
    }

    public Individual getLooser() {
        return looser;
    }

}
```

L'algorithme !

Tous les ingrédients sont en place, il ne manque maintenant que la traduction en Java du pseudo-code de l'algorithme donné en début d'article. La voici :

```
public class Optimiser {

    public static void run(int populationSize, int genomeSize) {

        Individual elite = null;
        Vector vector = new Vector(genomeSize);

        int iteration = 0;

        do {

            iteration++;
            System.out.println("Iteration " + iteration);

            Population population = new Population(populationSize, vector);
            population.evaluate();

            Individual winner = population.getWinner();
            Individual looser = population.getLooser();

            System.out.println("Winner is " + winner.getName() + " (fitness : " + winner.getFitness() + ")");
            System.out.println("Looser is " + looser.getName() + " (fitness : " + looser.getFitness() + ")");

            if (elite == null || elite.getFitness() < winner.getFitness()) {
                elite = winner;
                System.out.println("New elite " + elite.getName() + " with fitness " + elite.getFitness() + " !");
            } else {
                System.out.println(elite.getName() + " with fitness " + elite.getFitness() + " remains elite !");
            }
```

```
            System.out.println("Genome of elite is : " + elite.getGenome());

            vector.update(elite, looser);

        } while (iteration < Constants.MAX_ITERATIONS && vector.getConvergence() < Constants.
MAX_CONVERGENCE);

    }

}
```

Comme annoncé, l'algorithme est très simple : la moitié des lignes correspondent à des traces affichées sur la console afin de faciliter le suivi de son déroulement !
Ce code introduit cependant une nouvelle notion : celle d' « élitisme ». Tout comme pour les algorithmes génétiques, cette notion est optionnelle mais permet le plus souvent d'améliorer la convergence de l'algorithme. Est considéré comme « élite » l'individu le plus adapté rencontré jusqu'à présent, toutes populations confondues. Son génome va être utilisé pour la mise à jour de la distribution de probabilité, en lieu et place de celui de l'individu le mieux adapté de l'itération courante. Voici une trace de l'exécution de l'algorithme, pour les données d'entrée suivantes :

- Poids : { 61, 56, 99, 75, 13, 20, 17, 8, 9, 44 }
- Utilité : { 76, 31, 92, 37, 63, 78, 47, 17, 38, 48 }
- Total poids = 402
- Total utilité = 527
- Poids maximal = 265

```
Iteration 1
Fitness of 1-1 = 187
Fitness of 1-2 = -1
Fitness of 1-3 = -1
Fitness of 1-4 = 271
Winner is 1-4 (fitness : 271)
Looser is 1-2 (fitness : -1)
New elite 1-4 with fitness 271 !
Genome of elite is : 0011011100
Convergence = 0.14500001
Iteration 2
Fitness of 2-1 = 278
Fitness of 2-2 = 265
Fitness of 2-3 = 238
Fitness of 2-4 = 229
Winner is 2-1 (fitness : 278)
Looser is 2-4 (fitness : 229)
New elite 2-1 with fitness 278 !
Genome of elite is : 1010101000
Convergence = 0.15975001
Iteration 3
Fitness of 3-1 = 157
Fitness of 3-2 = 207
Fitness of 3-3 = 421
Fitness of 3-4 = 170
Winner is 3-3 (fitness : 421)
Looser is 3-1 (fitness : 157)
New elite 3-3 with fitness 421 !
Genome of elite is : 1010111101
Mutation bit 6, old = 0.69371873, new = 0.7090328
Convergence = 0.2532425
(...)
Iteration 71
Fitness of 71-1 = 421
Fitness of 71-2 = 421
Fitness of 71-3 = 421
Fitness of 71-4 = 421
Winner is 71-1 (fitness : 421)
Looser is 71-1 (fitness : 421)
3-3 with fitness 421 remains elite !
Genome of elite is : 1010111101
Convergence = 0.9900888
```

Et voici les paramètres par défaut utilisés pour contrôler la convergence et l'arrêt de l'algorithme :

```
public class Constants {

    public static final float LEARN_RATE = 0.1f;
    public static final float NEGATIVE_LEARN_RATE = 0.075f;
    public static final float MUTATION_PROBABILITY = 0.02f;
    public static final float MUTATION_SHIFT = 0.05f;

    public static final int MAX_ITERATIONS = 9999;
    public static final float MAX_CONVERGENCE = 0.99f;

}
```

CONCLUSION

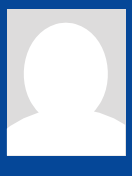
Tout comme les algorithmes génétiques, l'apprentissage incrémental à population s'inspire des théories évolutionnistes qui utilisent la notion d'adaptation (fitness) pour converger au fil des générations (itérations) vers un individu (une solution) le plus adapté à son environnement (une solution optimale). A la différence de ces derniers, les opérations de sélection et de croisement génétique sont cependant abandonnées, rendant ainsi plus simple l'implémentation de l'algorithme. A vos claviers !

Pour aller plus loin…

Les lecteurs pourraient se demander si le codage binaire des gènes retenu dans cet article n'est pas un facteur limitant, réduisant fortement l'intérêt de l'algorithme. Il n'en n'est rien ! Il est tout à fait possible de combiner plusieurs bits adjacents pour former un gène pouvant prendre plusieurs valeurs distinctes (8 bits formant ainsi un ensemble de 256 valeurs possibles). Attention toutefois : si l'on retient l'encodage binaire classique de valeurs entières (0 = '0000', 1 = '0001', 2 = '0010', …, 15 = '1111'), la convergence de l'algorithme risque d'être erratique. En effet, passer par exemple de la valeur 7 ('0111') à la valeur 8 ('1000') nécessite de modifier 4 bits, et autant de probabilités associées... Pour cela, il existe une solution simple : utiliser un encodage binaire ne modifiant qu'un bit à la fois lors des opérations d'incrémentation ou de décrémentation. C'est ce que permet par exemple le code de Gray ([3]) ! •

Liens

[3] https://fr.wikipedia.org/wiki/Code_de_Gray



Eugene Shen, Vincent Bechu,
François Le Corre et Sven Franck

Partie 1

OfficeJS : réaliser une Progressive Offline Web Application simple avec RenderJS et jIO

OfficeJS est une suite bureautique incluant plusieurs applications HTML5 : traitement de texte, présentation, tableur, illustration, traitement d'images, etc. La suite est compatible avec les principaux standards bureautiques et fonctionne en mode hors ligne (mode offline). L'appstore d'OfficeJS est aussi le cœur de la nouvelle interface d'ERP5, un ERP/CRM libre. Ces solutions sont publiées par Nexedi. Aujourd'hui, nous allons expliquer comment créer une application HTML5 avec OfficeJS, qui fonctionne sans connexion réseau en utilisant le concept du Progressive Offline Web Application (POWA).

Nous utiliserons, pour le front-end, RenderJS, une bibliothèque JavaScript créée par Nexedi. Pour le back-end, nous utilisons jIO, un autre composant JS. Il fournit une API pour implémenter les différents services de stockage en ligne (DropBox, GDrive, WebDAV, ERP5, etc.) ou hors ligne (IndexedDB, LocalStorage, WebSQL). Après une introduction de RenderJS et jIO, nous suivrons la même démarche que celle du site TodoMVC et créerons une version simple d'un gestionnaire de tâches comme TodoMVC.

Pour bien commencer

Une bonne connaissance de JS est nécessaire pour bien maîtriser cet article. Ce que nous appellerons par la suite "gadget" est constitué de fichiers HTML, JavaScript et CSS. Il doit pouvoir fonctionner de manière autonome et pouvoir être intégré dans un gadget parent, placé dans une iframe, ou directement dans le DOM.

Nous commencerons par créer un dossier dans lequel nous placerons les fichiers RenderJS, jIO, et RSVP.js, de préférence nommés renderjs.js, jio.js et rsvp.js respectivement. La première application que nous allons créer étant plutôt simple, nous placerons tous les fichiers suivants dans le même dossier, mais ce n'est en aucun cas obligatoire.

"Hello World"

Enregistrez le code HTML suivant dans un fichier index.html.

```
<!doctype html>
<html>
 <head>
  <title>OfficeJS App</title>
  <script src="rsvp.js"></script>
  <script src="renderjs.js"></script>
  <script src="jio.js"></script>
  <script src="index.js"></script>
  <link href="index.css" rel="stylesheet">
 </head>
 <body>
  <h1>OfficeJS Application</h1>
  <p></p>
 </body>
</html>
```

jIO et RenderJS dépendent de RSVP et doivent être chargés après ce dernier. index.html doit contenir les fichiers index.css

```
h1 {
 color: red;
}
```

et index.js respectivement

```
(function (window, rJS) {
 rJS(window)
  .declareService(function () {
   this.element.querySelector("p").textContent = "Hello, world!";
  });
}(window, rJS));
```

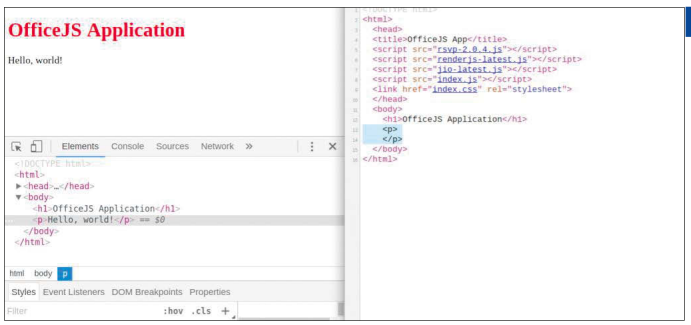
Le code JavaScript est enrobé dans une fonction immédiatement exécutée qui prend en paramètre rJS et window. Le mot-clef this correspond au gadget dans lequel il est appelé. Le gestionnaire d'évènement declareService() se déclenche dès que le gadget est chargé dans le DOM, rajoutant le texte "Hello World" à l'élément sélectionné. 1

Notre première app RenderJS est terminée. Pour voir le résultat, il suffit d'ouvrir index.html dans le navigateur.

Plus de fonctionnalités

Rajoutons les lignes suivantes dans le <body> de index.html pour pouvoir jouer avec plus d'éléments :

```
<form>
 <input type="text">
```



1

```
</form>
<ul></ul>
```

Rajoutons des évènements JS pour les éléments rajoutés dans le html. Modifiez votre fichier JavaScript pour qu'il ressemble à ça :

```
(function (window, document, rJS) {
 rJS(window)
  .declareService(function () {
   this.element.querySelector("p").textContent = "Hello, world!";
  })
  .declareMethod("addItem", function (item) {
   var list_item = document.createElement("LI");
   list_item.appendChild(document.createTextNode(item));
   this.element.querySelector("ul").appendChild(list_item);
   this.element.querySelector("p").textContent =
    "Added the new item, " + this.state.current_item +
    ". Total " + this.state.item_list.length + " item(s).";
  })
  .onEvent("submit", function (event) {
   var item = event.target.elements[0].value;
   event.target.elements[0].value = "";
   this.addItem(item);
  }, false, true);
}(window, document, rJS));
```

L'objet document est rajouté dans le scope pour pouvoir y accéder depuis une méthode personnelle, créée par declareMethod(). Cette méthode ajoute des éléments textuels à une liste. Ce texte est créé par un "event listener" lié à submit par onEvent(). Rafraîchissez votre navigateur en nettoyant le cache, tapez une phrase dans le formulaire puis soumettez-le en appuyant sur la touche Entrée. Une liste se crée puis se remplit en fonction de ce que vous envoyez depuis le formulaire. Vous venez de faire votre première méthode RenderJS.

Plus de méthodes

Vous pouvez rajouter des méthodes RenderJS où vous voulez dans votre chaîne d'évènements. Nous allons ajouter deux autres méthodes à notre fichier index.js:

```
.setState({
 item_list: [],
 current_item: null
})
.onStateChange(function (modification_dict) {
 if (modification_dict.hasOwnProperty("current_item")) {
  this.addItem(modification_dict.current_item);
  this.state.item_list.push(modification_dict.current_item);
 }
})
```

Modifions ensuite la manière dont notre écouteur d'évènement fonctionne, en remplaçant this.addItem(item) par

```
this.changeState({current_item: item});
```

dans la méthode .onEvent("submit", function).

La méthode setState() est appelée automatiquement, comme declareService(), et initialise l'état du gadget à l'objet passé en paramètre. La méthode onStateChange() se déclenche si et seulement si changeState() est appelé avec un état différent de celui en cours. Attention, seules les réassignations sont prises en compte, les copies ne le sont pas. Après avoir rafraîchi l'application, vous pouvez tester en appuyant plusieurs fois sur entrée sans modifier la valeur dans le formulaire. Aucun nouvel élément n'est ajouté excepté le premier. Notons d'abord que changeState() n'écrase pas l'état actuel, il change seulement les propriétés que l'on a passées en argument. Ainsi pour retirer une propriété de l'objet créé par setState(), il faut lui passer cette propriété avec la valeur undefined.

Programmation Asynchrone

RenderJS fonctionne de manière totalement asynchrone, en utilisant les promesses implémentées par RSVP.js. Cependant, à la place d'utiliser .then() pour créer des chaînes de promesses, nous avons customisé une version de RSVP.js en y ajoutant une file d'attente RSVP.Queue(), qui permet d'annuler la chaîne de promesses. Ajoutons-le à notre scope :

```
(function (window, document, RSVP, rJS) {
 //...
}(window, document, RSVP, rJS));
```

Les méthodes de RenderJS renvoient toujours des chaînes de promesses. Les promesses ne se résolvent que quand elles ont fini d'être exécutées. Nous venons de voir qu'une file d'attente RSVP.Queue() existe, essayons de l'utiliser dans la méthode .onEvent("submit"):

```
.onEvent("submit", function (event) {
 var gadget = this;
 var item = event.target.elements[0].value;
 event.target.elements[0].value = "";
 return new RSVP.Queue()
  .push(function () {
   return gadget.changeState({current_item: item});
  })
  .push(function () {
   gadget.element.querySelector("p").textContent =
    "Added the new item, " + gadget.state.current_item +
    ". Total " + gadget.state.item_list.length + " item(s).";
  });
}, false, true);
```

N'oubliez pas de nettoyer l'update de ("p").textContent dans addItem. Attention, le this dans les méthodes de rJS(window) ne se réfère pas à la même chose que le this des méthodes de RSVP.Queue().
Il faut l'enregistrer dans une variable, gadget par convention, pour garder accès à notre gadget pendant une file de promesses.
Si vous rafraîchissez la page, tout doit fonctionner correctement. Seulement, la façon de régler ce problème n'est pas satisfaisante.
En effet, nous savons maintenant que toutes les méthodes de RenderJS retournent des files de promesses, changeState() ne faisant pas exception. Il devrait donc être possible de chaîner des promesses derrière changeState(), et avoir ainsi un résultat plus compact et plus lisible. Essayons donc de modifier onEvent():

```
.onEvent("submit", function (event) {
 var gadget = this,
  item = event.target.elements[0].value;
 event.target.elements[0].value = "";
 return gadget.changeState({current_item: item})
  .push(function () {
   gadget.element.querySelector("p").textContent =
    "Added the new item, " + gadget.state.current_item +
    ". Total " + gadget.state.item_list.length + " item(s).";
  });
}, false, true)
```

Rafraichissez, et voilà : 2
Une liste complète des méthodes se trouve en anglais à l'adresse https://renderjs.nexedi.com/

"TodoMVC"

Jusqu'à présent, nous n'avons travaillé qu'avec un gadget qui gère ce qu'on voit et ce qu'on peut faire sur l'application. Pour créer une application de gestion de tâche fonctionnelle, à la manière de l'application TodoMVC, il nous manque ce qui va communiquer avec notre base de données : le modèle. Créons les fichiers gadget_model.html et gadget_model.js:

```
<!doctype html>
<html>
 <head>
  <title>Model Gadget</title>
  <script src="rsvp.js"></script>
  <script src="renderjs.js"></script>
  <script src="jio.js"></script>
  <script src="gadget_model.js"></script>
 </head>
 <body>
 </body>
</html>

(function (window, rJS) {
 rJS(window)
  .declareService(function () {
   console.log("Hello, world!");
  })
}(window, rJS));
```

Ce gadget ne possède ni feuille de style, ni body : un modèle ne fait que modéliser l'information et ne l'affiche pas. Cependant, il faut y inclure tout de même les sources JavaScript : chaque gadget doit pouvoir être lancé de manière autonome, RenderJS s'occupant de charger chaque script présent dans les header une unique fois. Le nouveau gadget s'intègre dans une du corps du fichier index.html :

```
<div data-gadget-url="gadget_model.html"
 data-gadget-scope="model"
 data-gadget-sandbox="public">
</div>
```

L'attribut data-gadget-url va chercher le gadget à l'URL donnée, data-gadget-scope est un identifiant unique pour le gadget et data-gadget-sandbox

contrôle la manière d'intégrer le gadget dans l'application : public pour le mettre dans le DOM et iframe permettant à RenderJS de créer une iframe pour ce gadget. Notez, que RenderJS utilise XMLHttpRequest pour charger les autres gadgets comme gadget_model.html.

Les Bases de jIO

jIO est une bibliothèque fournissant une API unique pour différentes bases de données. Nous travaillons alors avec des documents et des attachements, qui sont respectivement des objets JSON et des binaires (Blob). Chaque document possède un identifiant unique. Remplaçons maintenant le code de gadget_model.js par le code suivant :

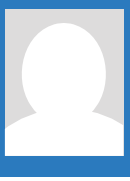
```
/*global window, RSVP, rJS, jIO*/
(function (window, RSVP, rJS, jIO) {
 "use strict";
 rJS(window)
  .declareService(function () {
   return this.changeState({
    storage: jIO.createJIO({
     type: "indexeddb",
     database: "todos-renderjs"
    })
   });
  })
  .declareMethod("post", function () {
   return this.state.storage.post.apply(this.state.storage, arguments);
  })
  .declareMethod("get", function () {
   return this.state.storage.get.apply(this.state.storage, arguments);
  })
  .declareMethod("allDocs", function () {
   return this.state.storage.allDocs.apply(this.state.storage, arguments);
  });
}(window, RSVP, rJS, jIO));
```

Ici, nous utilisons le constructeur createJIO() qui crée une interface jIO et l'assigne à this.state.storage. C'est par cette interface que l'on accédera à nos données. Contrairement aux autres méthodes jIO qui sont asynchrones, createJIO() est synchrone.
L'API de jIO est composée des méthodes suivantes :

- Pour les documents, nous disposons des méthodes post(), put(), get(), remove() et allDocs() ;
- Pour les attachments, putAttachment(), getAttachment(), removeAttachment() et allAttachments().

Pour plus d'informations : https://jio.nexedi.com/

Suite le mois prochain

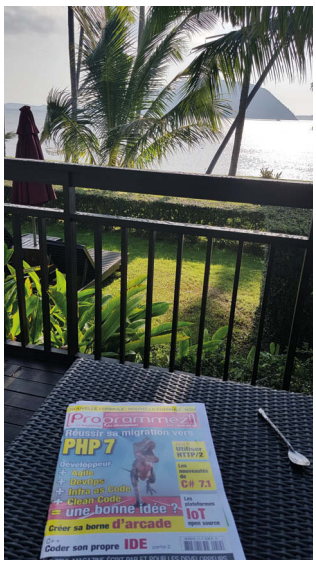


Richard Clark
Consultant indépendant .NET, MVP depuis 2002. Il anime le site c2i.fr et participe au podcast consacré aux technologies .NET devapps.be.

Amusons-nous avec BabylonJs, Angular et Microsoft Graph

J'ai pris l'habitude de prendre quelques jours de congés entre Noël et le jour de l'an. Et que fait un développeur pendant ses vacances ? Eh bien il développe, mais pour s'amuser.

Cette année, je venais de m'acheter un casque de réalité virtuelle, le Lenovo Explorer et le petit papa Noël m'avait gentiment apporté une caméra 360 de la marque Ricoh, la Theta V. D'où l'idée géniale (si, si, elle est géniale), de développer une application de réalité virtuelle.



Note : je présente ici les « grands » principes de l'application sans entrer dans les détails des implémentations. Le code complet est disponible sur mon compte Github : https://github.com/RichardC64.

J'adore les nouveaux bureaux de Programmez !

Première question : quelle techno utiliser ?

Dès qu'on parle de VR, la première technologie qui vient à l'esprit c'est bien entendu Unity. C'est d'ailleurs pourquoi Nicolas Sorel de Magma Mobile a très rapidement porté une de ses applications Android sur la plateforme Windows Mixed Reality parce que l'ensemble de ses jeux sont développés avec Unity. Mais bon, cela demande d'apprendre un logiciel et moi ce que je veux, c'est développer. Donc je me suis tourné vers le projet Open Source de Microsoft : BabylonJs.

Cela tombe bien car je connais bien son créateur, David Catuhe (je me souviens encore d'une discussion dans la salle d'embarquement de l'aéroport de Seattle en 2003 où il m'expliquait qu'il utilisait les GPU de ses cartes graphiques pour effectuer des calculs matriciels. En 2003 ! Un vrai fou furieux je vous dis).

Ensuite, pas question de programmer en Javascript mais en Typescript.

Puis je me dis que mon application risque de ne pas être qu'une application VR à 100%, mais intégrée au sein d'une application web plus large. Comme je fais de l'Angular à longueur de journée en ce moment, va pour Angular avec comme outil Angular CLI.

Enfin, l'outil de développement est naturellement Visual Studio Code. On résume donc le ticket gagnant :

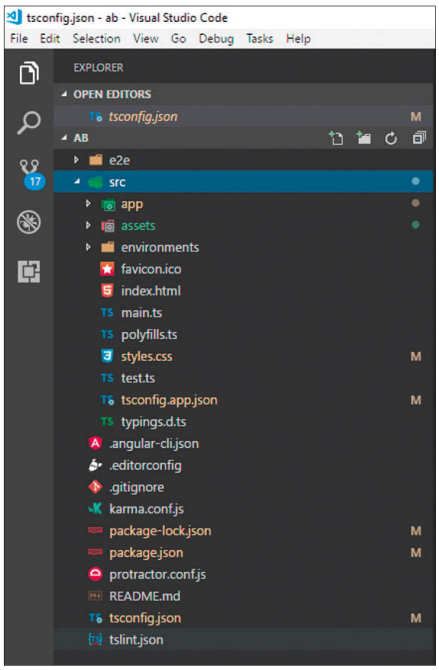
Angular (+CLI), BabylonJs, Typescript, Visual Studio Code.

Mise en place du projet

Nous supposerons ici que vous maitrisez Angular CLI. Cet utilitaire vous permet de créer très rapidement la structure de votre application Angular. Un petit :

```
ng n ab
```

En ligne de commande et hop, le projet est créé et l'on peut le charger dans Visual Studio Code.



Notre projet comportera dans un premier temps une page d'accueil (home) puis une page viewer. La génération des deux composants correspondants est effectuée par Angular CLI :

```
ng g c home
ng g c viewer
```

Passons sur la mise en place de la navigation pour nous concentrer sur l'intégration de BabylonJs dans le composant Viewer.

La première étape consiste à ajouter une référence à ce moteur 3D dans le fichier package.json :

```
"dependencies": {
  "babylonjs": "^3.2.0-alpha0",
  "babylonjs-loaders": "^3.2.0-alpha0",
  "@angular/animations": "^5.0.0",
  "@angular/common": "^5.0.0",
  "@angular/compiler": "^5.0.0",
  "@angular/core": "^5.0.0",
  "@angular/forms": "^5.0.0",
  "@angular/http": "^5.0.0",
  "@angular/platform-browser": "^5.0.0",
  "@angular/platform-browser-dynamic": "^5.0.0",
  "@angular/router": "^5.0.0",
  "core-js": "^2.4.1",
  "rxjs": "^5.5.2",
  "zone.js": "^0.8.14"
},
```

Remarquez que j'ai ajouté aussi la référence à babylonjs-loader qui nous permettra de charger des modèles 3D comme par exemple les contrôleurs.

Le principe de babylonJs est d'effectuer le rendu 3D de votre scène dans un canvas Html5. Donc dans notre component viewer, nous allons ajouter juste un tag canvas :

```
<canvas id="renderCanvas" touch-action="none"></canvas>
```

Tout est en place pour créer la logique de notre jeu 3D. Pour cela, nous ajoutons une classe Typescript simple appelée Game. Elle gèrera le moteur 3D (Engine), la scène (Scene), les caméras (camera), lumières (light) et objets 3D (meshes).
Comme dans tout moteur 3D, il y a une première phase de création de la scène puis une boucle de rendu.

```
import {
Engine, Scene, Light,
Vector3, HemisphericLight, MeshBuilder,
ArcRotateCamera, StandardMaterial, Texture,
Color3
} from 'babylonjs';
export class Game {
    private canvas: HTMLCanvasElement;
    private engine: Engine;
    private scene: Scene;
    constructor(canvasElement: string) {
      this.canvas = <HTMLCanvasElement>document.getElementById(canvasElement);
        // on attache le moteur de rendu avec le canvas
        this.engine = new Engine(this.canvas, true);
        …
    }
    createScene(): void {
        // création de la scène associée au moteur de rendu (engine)
        …
    }
    run(): void {
        // boucle du rendu
        this.engine.runRenderLoop(() => { this.scene.render(); });
    }
}
```

Dans la boucle de rendu, on ajoute le rendu de notre scène (pour la supprimer, stopRenderLoop, logique non ?). Cette classe sera instanciée et utilisée dans notre composant Viewer :

```
import { Component, OnInit } from '@angular/core';
import { Game } from '../logic/game';
@Component({
    templateUrl: './viewer.component.html'
})
export class ViewerComponent implements OnInit {
    constructor() { }
    ngOnInit() {
        const game = new Game('renderCanvas'); // id du tag du canvas
        game.createScene(); // initialisation de la scène
        game.run(); // lancement de la boucle de rendu
    }
}
```

Il ne nous reste plus qu'à créer notre scène.

Entrons dans la danse

Le cœur est la création de la scène (je ne suis pas en train de faire un jeu avec de l'IA, de l'interaction poussée, etc.).
Il faut donc créer au moins :

- Une caméra (pour voir, c'est mieux),
- Une lumière (à moins que vous soyez nyctalope),
- Un objet/mesh à voir (même éclairé, du rien c'est du rien et ce n'est pas grand-chose).

Pour tous ces objets, BabylonJs nous propose des classes (Typescript) dont les constructeurs font généralement 99.9% du travail. Par exemple pour une lumière hémisphérique (on a toutes sortes de lumières : directionnelles, spots, etc.) :

```
this.light = new HemisphericLight('skyLight', new Vector3(1, 1, 0), this.scene);
```

Remarquez le 1er argument qui est le nom de votre lumière (qui permet de la retrouver, l'identifier plus tard), et le dernier est la scène dans laquelle on l'ajoute. On retrouve ce pattern un peu partout comme pour la création d'une caméra ou d'une texture.
On ajoute la caméra de la même façon et il ne nous reste plus que l'objet/mesh. Je vous rappelle que l'objectif premier était de pouvoir visualiser en VR mes photos 360° prise avec mon Theta V.
Pour cela, on va créer une boîte (ou une sphère si vous voulez) sur laquelle on va appliquer comme texture notre photo. Enfin, quand je dis on va créer une boîte, c'est le framework qui va faire tout le travail pour nous grâce au MeshBuilder et sa méthode CreateBox :

```
const skybox = BABYLON.MeshBuilder.CreateBox('skyBox', { size: 1000.0 }, this.scene);
```

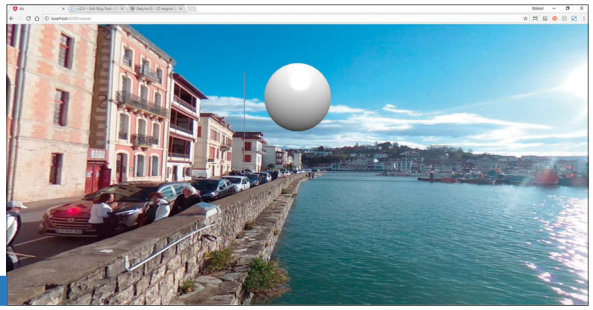
Simple non ? Je vous invite fortement regarder le code de BabylonJs (il est en Open Source) pour voir ce qu'il fait, ce n'est pas si simple. Si vous avez déjà fait de la 3D avec DirectX ou même XNA, vous saurez que ces moteurs de rendu sont basés sur des sommets (vertices), des triangles avec une orientation. Donc quand vous demandez de créer une boîte, il vous crée les 6 sommets, les 2 triangles par face, le mapping/colorimétrie des sommets, etc.
Mais pour nous, c'est une ligne de code !
Ensuite, on lui applique un matériau qui est composé d'une texture de l'image 360 :

```
const skyboxMaterial = new StandardMaterial('skyBox', this.scene);
skyboxMaterial.backFaceCulling = false;
// création de la texture de l'environnement
const fixedTexture = new Texture('/assets/R0010081.JPG', this.scene);
skyboxMaterial.reflectionTexture = fixedTexture;
skyboxMaterial.reflectionTexture.coordinatesMode = Texture.FIXED_EQUIRECTANGULAR_MODE;
skyboxMaterial.diffuseColor = new Color3(0, 0, 0);
skyboxMaterial.specularColor = new Color3(0, 0, 0);
skybox.material = skyboxMaterial;
```

Et voilà le travail : 1

Et la VR dans tout ça ?

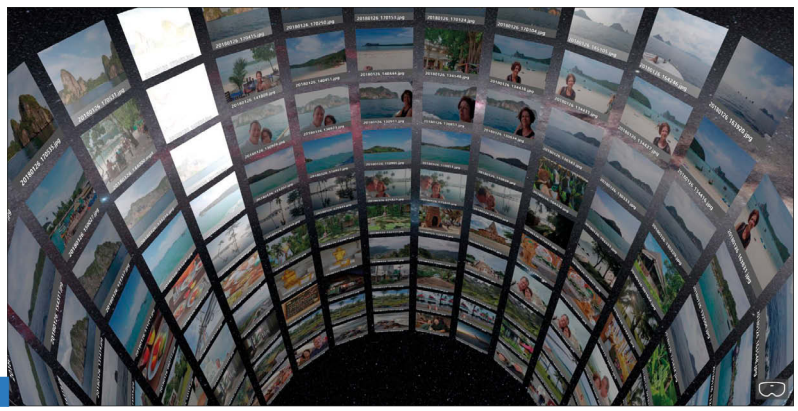
Vous allez me dire, il y a autant de VR dans ce que je viens de présenter que de côte de bœuf dans l'assiette d'une Végane !
Heureusement, depuis peu, BabylonJs vous permet de visualiser votre scène 3D avec votre casque de réalité virtuelle grâce à un helper dédié, le VRExperienceHelper et en 3 lignes de code :



1

NB : dans la scène ci-dessus j'ai ajouté une sphère blanche.

2



3

```
this.vrExperienHelper = this.scene.createDefaultVRExperience();
this.vrExperienHelper.enableInteractions();
this.vrExperienHelper.displayLaserPointer = true;
```

En ajoutant ces 3 lignes, vous verrez apparaître en bas à droite un bouton en forme de casque. Cliquez dessus, enfilez votre casque VR et … 2

Bon avec le casque, ça rend mieux : vous pouvez tourner la tête et admirer votre photo 360.

En cliquant sur ce bouton, il vous remplace votre caméra par une nouvelle caméra, une VRDeviceOrientationFreeCamera qui se charge de tout. En présentant ce petit projet à mes camarades de l'excellent podcast http://devapps.be, le sieur Christophe Peugnet me dit tout de go :

> *« Ce serait super si l'on pouvait voir toutes ses photos en VR. »*

Il est gentil comme garçon mais je n'ai qu'une semaine de repos entre Noël et le jour de l'an. Mais ni une, ni deux, je m'attèle tout de même à la tâche. Dans cette deuxième phase, je veux donc visualiser mes photos archivées dans OneDrive. Le résultat final (pour l'instant) est le suivant : 3

Les dossiers et les photos sont disposées le long d'un cylindre. Quand je clique sur un dossier, j'affiche son contenu. Et tourner la tête pour voir toutes ces photos d'un seul coup d'œil, je vous avoue que c'est vraiment sympathique.

Microsoft Graph

Il me faut donc accéder à la liste de mes photos sur OneDrive et pour cela j'ai à ma disposition Microsoft Graph. Cela tombe mal car je n'avais jamais touché à la bête. Heureusement, on retrouve des concepts qui existaient dans le Live SDK donc je n'étais pas en complet terrain inconnu.

Première étape : déclarer son application auprès de Microsoft Graph.

Allez sur https://developer.microsoft.com/fr-fr/graph , identifiez-vous puis ajoutez votre application (de type convergente). Le plus important est d'ajouter une url de redirection (http://localhost:4200/callback dans mon exemple). Attendez quelques minutes avant que ce soit vraiment actif (faut savoir être patient dans la vie).

L'application est prête à accéder aux informations du Microsoft Graph. Pour identifier l'utilisateur, il existe des composants Javascript qui font ce travail pour vous mais je voulais comprendre en détail son fonctionnement.

Le principe est le suivant :

Quand je clique sur le bouton Log In de ma page, j'ouvre une fenêtre popup. L'url de cette fenêtre est la page d'identification de Microsoft Online :

```
private applicationConfig = {
   clientID: 'guid de votre application',
   graphScopes: 'user.read files.read files.read.all sites.read.all'
 };
public getUrl(): string {
    let url = 'https://login.microsoftonline.com/common/oauth2/v2.0/authorize';
   url += '?client_id=' + this.applicationConfig.clientID;
   url += '&response_type=token';
   url += '&redirect_uri=' + encodeURIComponent('http://localhost:4200/callback');
   url += '&scope=' + encodeURIComponent(this.applicationConfig.graphScopes);
   return url;
 }
```

Dans l'url appelée on voit que l'on a bien l'id de votre application, les autorisations voulues (scopes) ainsi que l'url de la page (callback) qui sera appelée quand l'identification aura eu lieu.

L'utilisateur a devant les yeux la page d'identification de Microsoft classique. Il s'identifie et le site le redirige alors vers votre page (callback) avec dans l'url appelée, votre access token, la date d'expiration, les autorisations.

Exemple :

http://localhost:4200/callback#access_token=EwB...Q53d&token_type=bearer&expires_in=3600&scope=User.Read%20Files.Read%20Files.Read.All%20Mail.Send

Dans ma page callback, je lis ces informations et je les stocke dans le localstorage. Je rafraîchis la page appelante et je ferme ma fenêtre popup :

```
public onCallback() {
  const authInfo = this.getAuthInfoFromUrl(); // lecture des arguments de l'url
  const token = authInfo['access_token'];
  const expiry = parseInt(authInfo['expires_in'], null);
  if (token) {
   this.saveAuthToken(token, window); // service stockant dans le localstorage
  }
  window.opener.location.reload(); // on recharge la page appelante
  window.close(); // on ferme le popup
 }
```

J'ai donc maintenant dans le localstorage mon token que je vais envoyer avec chacune de mes requêtes vers les APIs de MS Graph (grâce à une classe dérivant de HttpInterceptor d'Angular).
Par exemple, pour recevoir la liste des driveItem d'un driveItem, j'ai créé un service qui appelle l'url suivante :

```
https://graph.microsoft.com/v1.0/me/drive/items/
   monDriveItemId
   ?$expand=thumbnails,children($expand=thumbnails($select=medium))';
```

Pour la racine de mon OneDrive, je peux utiliser 'root' comme valeur de monDriveItemId. L'appel de ces APIs me retourne un JSON complexe. Heureusement, Microsoft met à notre disposition un fichier de définition de type (.d.ts) que je peux référencer dans mon package .json pour bénéficier de l'IntelliSense :

```
"@microsoft/microsoft-graph-types": "^1.1.0"
```

Mon service me retourne ainsi un objet typé, un DriveItem avec une propriété children qui représente l'ensemble de ses enfants (les sous-dossiers et les fichiers).

Retour à ma scène 3D

En récupérant l'ensemble des enfants du driveItem, je dois construire dynamiquement ma scène, c'est-à-dire construire les éléments 3D du cylindre (cf. capture). En réalité, chaque élément est constitué de 3 plans : 4
Le premier est un plan avec comme texture la miniature du driveItem.
Le second est un plan bleu (pour un dossier), gris pour un fichier et le dernier est un plan avec le texte.
J'utilise pour les trois la méthode MeshBuilder.CreatePlane ; il n'y a que les textures qui changent. C'est relativement simple, il n'y a que le positionnement de ces plans dans l'espace qui m'a demandé un peu de réflexion (rotation + translation).
Une dernière chose : je ne rends « pickable » que le plan avec l'image et si ce dernier est un dossier. Ainsi, si l'on clique (pointe avec les contrôleurs VR) sur cette image, on rafraîchit le contenu de la scène 3D avec les sous-dossiers et fichiers du nouveau driveItem (avec ré-interrogation de l'API MS Graph).

4

Conclusion

Vous voyez, en une petite semaine, j'ai pu m'amuser avec BabylonJs et Microsoft Graph et obtenir un résultat ma foi fort sympathique. Il faudrait rajouter pas mal de « trucs » pour que cela se transforme en véritable application fonctionnelle, mais, déjà, on peut se balader en VR. Comme Microsoft entend supporter les applications PWA, on peut même envisager dans un avenir proche de créer ainsi très facilement des applications de réalités virtuelles disponibles sur le Microsoft Store. Si vous êtes intéressé, n'hésitez pas à me rejoindre dans le projet sur Github. Je tiens à remercier David Rousset pour ses conseils pendant le développement de cette application. Il est l'un des principaux artisans de BabylonJs avec David Catuhe et Etienne Margraff. •

Christophe Michel
Software Gardener

Écrire une bibliothèque en Java

Cette fois ça y est. Ce petit bout de code Java bien pratique que vous avez terminé récemment, vous aimeriez bien le partager avec le monde entier. Parce qu'il simplifie la vie, parce qu'il est différent de l'existant, ou meilleur, ou plus simple d'utilisation.

Quelle que soit votre motivation, ce que vous voulez, c'est écrire une bibliothèque en Java. La bonne nouvelle, c'est que ça reste le code Java dont vous avez l'habitude. Mais si votre job de tous les jours consiste à écrire ou maintenir des applications, il y a un certain nombre de différences notables entre le moment où ça compile pour la première fois et les millions (que dis-je, les milliards!) de téléchargements sur Maven Central.
Dans cet article, je partirai du principe que vous publierez votre bibliothèque en open source et vous présenterai la marche à suivre pour mener à bien ce type de projet.

Le code

Tant que votre code reste confiné dans votre projet, vous faites un peu ce que vous voulez. Dans les limites du raisonnable bien sûr, on n'est pas des bêtes, mais à partir du moment où vous ne distribuez pas le code, vous gardez une grande liberté d'écriture.

Dépendances

En revanche, lorsque vous commencez à être inclus dans les projets de tout un tas de monde, c'est une autre paire de manches!
A défaut d'écrire la bibliothèque sous forme de module (fonctionnalité introduite en Java 9), toutes les dépendances que vous intégrez dans votre bibliothèque seront transitivement présentes dans l'application de vos utilisateurs.
Si les versions que vous utilisez sont incompatibles avec les leurs, bienvenue dans le monde du jar hell et des exclusions de dépendances. Personne n'a envie de s'infliger ça, et trimbaler trop de dépendances dans votre bibliothèque risque de décourager autrui de s'en servir.
Les déboires encore récents des projets JavaScript reposant sur des myriades de micro-dépendances doivent nous rappeler que les limiter au maximum est une bonne idée dans n'importe quel projet, et une absolue nécessité pour une bibliothèque.

Logging

Un cas particulier de dépendances, c'est le logging. C'est une bonne idée d'en prévoir dans votre bibliothèque car bien conçu, il sera d'une aide précieuse à vos utilisateurs (et pourra vous servir aussi au passage).
Sauf que le framework de logging, c'est une dépendance, et qu'il serait assez inconvenant d'imposer le vôtre aux utilisateurs de votre bibliothèque. Le problème, c'est qu'à moins de vous restreindre à utiliser `java.util.logging`, vous allez bien devoir écrire votre code de logging avec un framework.

Heureusement, vous n'êtes pas le premier à rencontrer ce cas de figure et la solution est simple : utiliser une façade de logging. La référence actuelle est **SLF4J**, qui peut s'interfacer avec la plupart des implémentations populaires, telles que **Log4J** ou **Logback**. Elle se branchera sans problème sur celle de vos utilisateurs.

Injection de dépendance

La question de l'injection de dépendance est un peu plus délicate. Il est bien évidemment hors de question d'embarquer un framework "lourd" à la Spring pour assembler les classes de votre bibliothèque. Il existe toute une gamme de frameworks plus simples, mais dans le cadre d'un projet de ce type, il est tout à fait possible de s'en passer en revenant aux bonnes vieilles méthodes : des Factory et des Builder.
Veillez si possible à préserver l'extensibilité de votre bibliothèque en évitant les factory method statiques qui ne permettent pas de dériver les classes de votre API publique. Utilisez-les plutôt pour certaines dépendances techniques qu'il sera facile de d'instancier à part.

```
public class MyApi {

  // Méthode statique impossible à étendre
  public static MyApi getInstance() {
    return new MyApi();
  }

  private MyApi() {
    // Initialisation ...
  }

}

MyApi api = MyApi.getInstance();
```

Ici, sous-classer `MyApi` est possible, mais vous perdez le code d'initialisation, ce qui risque de poser problème. On préférera la construction plus standard, si possible en injectant un objet de configuration.

```
public class MyApi {

  private final MyApiConfiguration configuration;
```

```
  public MyApi() {
    this(MyApiConfiguration.getDefault());
  }

  public MyApi(MyApiConfiguration configuration) {
    this.configuration = configuration;
    // Initialisation
  }

}

MyAPI api = new MyApi();
```

La compatibilité

En écrivant une bibliothèque, vous allez distribuer du code et perdre le contrôle sur son utilisation. Vos utilisateurs vont étendre vos classes, s'en servir de façon imprévue et ne mettront pas forcément à jour le package quand vous le souhaiterez.

L'idéal pour une bibliothèque est d'utiliser **SemVer**, la gestion sémantique de versions. Le principe général de SemVer est le suivant :

- **Un changement de version majeure indique des modifications non rétrocompatibles de votre API publique ;**
- **Un changement de version mineure indique des modifications rétrocompatibles de votre API publique ;**
- **Un changement de version corrective indique des corrections d'anomalies rétrocompatibles.**

Dans le cadre d'une bibliothèque Java, la compatibilité s'entend au sens de la **compatibilité binaire**. Si une version de votre bibliothèque est rétrocompatible avec une version antérieure, le code continue de compiler, mais aussi de linker et de s'exécuter sans erreur.

Par exemple, renommer un champ privé ou rendre une méthode privée publique sont des changements rétrocompatibles. Supprimer une méthode publique ou changer un nom de package ne le sont pas. La documentation de la JVM dresse une liste exhaustive de ces changements.

Si jamais vous désirez que la cohabitation entre deux versions majeures de votre bibliothèque soit possible, vous devrez changer les noms de package Java et le Group Id Maven pour éviter toute collusion. C'est par exemple ce qu'a fait la bibliothèque **Jackson** en passant de org.codehaus.jackson à com.fasterxml.jackson.

La licence

Il est possible de publier son code ou ses binaires sans licence, mais si vous le faites, c'est le droit basique du copyright qui s'applique et dans ce cas **personne ne pourra utiliser votre création sans votre autorisation expresse**. Autant dire que c'est un frein conséquent pour un utilisateur.

Il existe une large gamme de licences pour l'open source, qui couvrent de nombreuses sensibilités et de nombreuses nuances. Le choix est un peu difficile pour le béotien mais heureusement on nous mâche régulièrement le travail avec des assistants de sélection comme **celui de GitHub**.

L'outillage

Gestion de version

Partons du principe que vous utilisez ***git***. Les plateformes les plus populaires **GitHub**, **GitLab** et **Bitbucket**. Pour un projet personnel open source, les trois sont relativement proches, à quelques différences près :

- **Bitbucket et GitLab proposent des dépôts privés dans leur formule gratuite ;**
- **GitHub dispose d'une communauté open source très importante et d'intégration avec un plus grand nombre de services tiers que ses concurrents.**

Intégration continue

Le choix de l'outil d'intégration continue, à moins de l'héberger soi-même, dépend en partie du choix de la solution de gestion de version.

Si vous avez opté pour GitLab ou BitBucket, ces services proposent des solutions d'intégration continue (**GitLab CI/CD** et **BitBucket Pipelines** respectivement) et le plus simple reste de les utiliser.

Si vous avez opté pour GitHub, le choix le plus populaire est **Travis**, auquel vous vous connecterez directement avec votre compte GitHub pour un accès immédiat à vos dépôts.

Travis est piloté par un fichier de configuration .travis.yml à placer à la racine du projet. Pour un projet Java 8 simple, le contenu est minimal :

```
language: java
jdk: oraclejdk8
cache:
  directories:
    - $HOME/.m2
```

La mise en cache du dépôt Maven local est vitale pour éviter des temps de build excessifs.

Une fois le projet configuré et buildé, vous pouvez modifier le README.md du projet pour **afficher sur sa page d'accueil un badge qui indique le statut du build**.

Qualité de code

Outre la possibilité d'embarquer des plugins de type **FindBugs** ou **PMD** directement dans le build, il existe des outils en ligne pour obtenir des indicateurs sur la qualité de code du projet.

Le plus populaire de tous est **SonarQube** qui s'interface avec un très grand nombre d'usines d'intégration continue et propose une version en ligne gratuite pour les projets open source.

Si vous avez choisi la combinaison GitHub/Travis, la mise en place est extrêmement simple (là encore, il est possible d'utiliser son compte GitHub pour se connecter à SonarQube) et elle est détaillée dans la **documentation de Travis**.

Si vous avez opté pour d'autres services, ils disposent généralement de plugins ou d'intégrations dédiés à Sonar.

Si vous n'êtes intéressé que par la couverture de tests, **Coveralls** est une bonne alternative. Il s'intègre actuellement avec les projets GitHub et Bitbucket, le support GitLab étant prévu à l'avenir.

Publication sur Maven Central

La publication de votre jar sur **Maven Central** pourrait faire l'objet d'un article à part entière. Il y a un certain nombre d'étapes à respecter, en voici un aperçu :

- **Inscrire votre projet via un ticket JIRA chez Sonatype ;**
- **Générer une clé GPG et paramétrer la signature automatique de vos livrables dans le build ;**
- **Déployer sur Maven Central.**

Pour le guide complet, je vous renvoie vers **le guide de Sonatype** (qui héberge le dépôt) et plus spécifiquement **le processus de déploiement avec Maven**.

Idéalement, vous voudrez que la génération des jars de code source, de Javadoc et la signature GPG des livrables soient des étapes associées à un profil de release qui n'est exécuté que lorsque vous effectuez la release. Non seulement vous gagnez du temps sur vos builds, mais cela vous évite une étape inutile lorsque vous faites de l'intégration continue, sur Travis par exemple.

Documentation

La documentation est importante dans n'importe quel projet de développement, mais elle est absolument vitale pour une bibliothèque.

Non seulement vous devrez vous assurer que le code lui-même est correctement documenté (que ce soit via une documentation explicite type Javadoc, un nommage approprié ou encore les tests) mais vous devrez également fournir une documentation d'utilisation.

"After just a few user testing sessions, it became clear that developers expected to learn how to work with a new SDK from examples."— How I do Developer UX at Google.

En effet, une des premières choses qu'un développeur va regarder avant de choisir d'utiliser une bibliothèque, c'est la façon dont elle s'utilise. Les exemples se doivent d'être clairs et abondants.

Écrivez une section "Quick Start" qui permet de commencer tout de suite à utiliser votre code sur un cas simple et n'hésitez pas à donner des cas plus détaillés.

Donnez les moyens aux développeurs de savoir rapidement et de façon fiable si l'outil que vous proposez leur est adapté. Il est souvent plus rapide de chercher une bibliothèque alternative à la vôtre que de passer trop de temps à comprendre ce qu'elle peut offrir ou comment elle fonctionne.

Contributions au projet

Puisque votre projet est open source, vous souhaitez peut-être que d'autres personnes contribuent au projet. Dans ce cas, vous devrez fournir des instructions claires sur le processus de contribution. Ces instructions incluent par exemple :

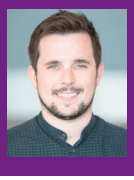
- **Comment soumettre une modification de code ;**
- **Comment signaler un bug ;**
- **Comment contacter le mainteneur du projet ;**
- **Où trouver la roadmap s'il y en a une.**

GitHub fournit **un guide pour faciliter les contributions à un projet** qui couvre ces aspects et bien d'autres encore. Certains font même l'objet de fonctionnalités spécifiques sur GitHub, qu'on peut également trouver chez ses concurrents.

Lancez-vous !

A ce stade, vous avez toutes les informations et pistes nécessaires pour démarrer votre projet. Reste quelque chose que l'on peut difficilement faire à votre place, c'est trouver une bonne idée et avoir la discipline de travailler dessus régulièrement.

Qui sait, vous créerez peut-être un futur élément incontournable de l'écosystème Java ? Et sans aller jusque-là, un projet de ce type est une bonne façon de démontrer vos compétences à un client ou un employeur potentiel. •

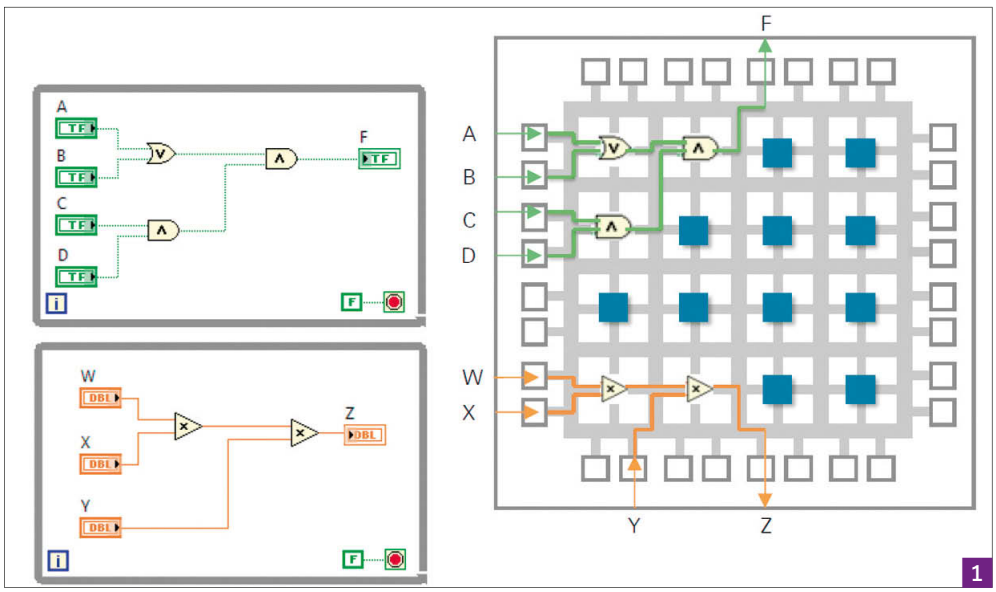


Antonin Goude
Ingénieur Produits Systèmes Embarqués,
National Instruments
antonin.goude@ni.com

Découverte de LabVIEW FPGA :
une approche différente pour le développement de systèmes embarqués

La conception de systèmes embarqués se définit toujours par l'association d'éléments matériels et de briques logicielles dans l'objectif d'exécuter une fonction prédéfinie. Sa particularité réside souvent dans ses capacités à résoudre des problèmes complexes et critiques. Lorsque ces systèmes remplissent cette fonction, on parle de systèmes déterministes ou temps réel. Afin de répondre à ces défis et aux contraintes des applications embarquées complexes, ces systèmes sont composés d'un ensemble d'éléments électroniques comme des microcontrôleurs, des DSP (Digital Signal Processor) ou encore de circuits logiques programmables, encore appelés FPGA. D'un point de vue logiciel, différents langages et outils de développement permettent alors de programmer et de configurer ces cibles, qui présentent chacun leurs propres avantages et inconvénients.

Le premier FPGA est apparu en 1985, grâce au travail de la société Xilinx. Ce type de composant a la particularité de combiner les avantages des ASIC traditionnels et de systèmes basés processeur. Ils offrent un cadencement matériel qui assure une vitesse élevée de fonctionnement et une grande fiabilité des tâches qui leur incombent. Mais contrairement aux processeurs, les FPGA connaissent un parallélisme natif, ainsi plusieurs opérations peuvent s'exécuter conjointement sans concurrence au niveau des ressources. Chaque tâche est affectée à une portion dédiée du circuit, et peut donc s'exécuter en toute autonomie. En conséquence, vous pouvez accroître le volume de traitement effectué sans que les performances d'une partie de l'application n'en soient affectées pour autant(1).



1

Définition des composants d'un FPGA

Malgré tout, chaque circuit FPGA est constitué d'un nombre fini de ressources, dont des matrices d'interconnexions programmables et des blocs d'entrées/sorties pour permettre au circuit de communiquer avec d'autres éléments du système comme des capteurs ou des actionneurs. Le FPGA est aussi et surtout composé d'éléments logiques configurables : des bascules (*flip-flops* en anglais) et des tables de correspondance (LUT ou *Look-Up Table* en anglais). Il s'agit de l'unité logique d'un FPGA. Leur interconnexion définira la fonctionnalité du circuit, du simple programme *Hello World* « matériel » au système de régulation PID par exemple. Associés à ces éléments logiques on retrouve également des blocs prédéfinis comme des mémoires RAM ou des multiplicateurs et des blocs DSP.

« Programmer » un FPGA

Il existe habituellement deux options principales pour programmer un FPGA, le Verilog d'une part, et le VHDL d'autre part. Il s'agit de langages de description de matériel (HDL pour *Hardware Description Language* en anglais) devenus rapidement indispensables pour développer sur ces cibles. Ces langages présentent des similitudes avec d'autres langages textuels, ce qui au premier abord permet de les prendre en main rapidement. Ils prennent aussi en considération le fait que, sur un FPGA, c'est l'utilisateur qui va définir l'architecture du circuit. Cela exige alors que les signaux soient connectés à partir de ports d'E/S externes sur des signaux internes qui, à leur tour, sont câblés aux fonctions qui contiennent les algorithmes. Ces fonctions s'exécutent de manière séquentielle et peuvent éventuellement être liées à d'autres parties du FPGA. L'ensemble des tâches qui s'exécutent en parallèle, comme nous l'avons vu précédemment, est toutefois difficile à visualiser dans un flux séquentiel ligne par ligne.

Pour vérifier le code créé, il est important de le tester dans un environnement de simulation. Cette phase permet de reproduire le cadencement du FPGA et l'ensemble des signaux d'entrée/sortie. Remarque impor-

(1) : http://www.ni.com/white-paper/6983/fr/

tante, cette étape est très souvent plus longue que la création du code lui-même ! Une fois vérifiée, la logique est transférée à un compilateur qui viendra la synthétiser sous forme de fichier de configuration ou *bitfile* qui nécessite souvent un travail manuel et parfois fastidieux de nommage des voies et des signaux. L'ensemble de ces processus requiert des connaissances avancées, ce qui explique que FPGA n'est pas toujours privilégié dans les conceptions de systèmes embarqués et ce malgré ces réels avantages.

Afin de rendre plus accessible la « programmation » d'un FPGA, des outils plus haut niveau comme LabVIEW, dits langages d'abstraction, sont apparus. L'environnement de programmation LabVIEW est particulièrement adapté à ce type de programmation car il permet de représenter graphiquement le parallélisme des tâches et le flux de données entre éléments logiques. Il permet par ailleurs d'associer du code HDL à du code graphique pour faciliter la réutilisation de code existant. Mais revenons tout d'abord à une rapide présentation de LabVIEW. 1

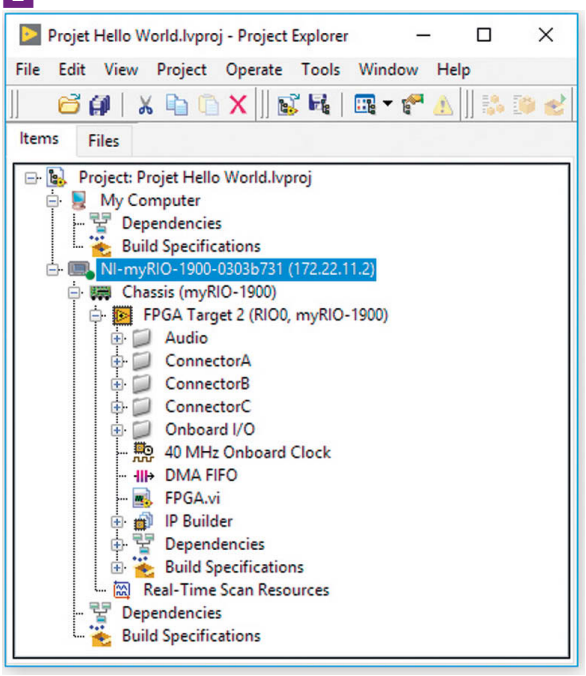
LabVIEW ? LabVIEW FPGA ? Kézako ?

Il s'agit d'un langage graphique basé sur le principe du flux de données, c'est-à-dire qu'il utilise des fils pour véhiculer les variables du programme d'une fonction à une autre. L'avantage principal de ce langage réside dans sa nature intuitive et la rapidité avec laquelle il permet d'obtenir une application fonctionnelle. L'IDE permet ainsi de programmer indépendamment des cibles de déploiements : PC, cibles National Instruments (myRIO, CompactRIO…) et même d'autres cibles matérielles comme Arduino ou Raspberry Pi. Aujourd'hui, nous allons nous focaliser sur des cibles à base de FPGA. Ainsi, le module LabVIEW FPGA est l'outil qui permet de concevoir graphiquement la logique nécessaire pour programmer les cibles FPGA.

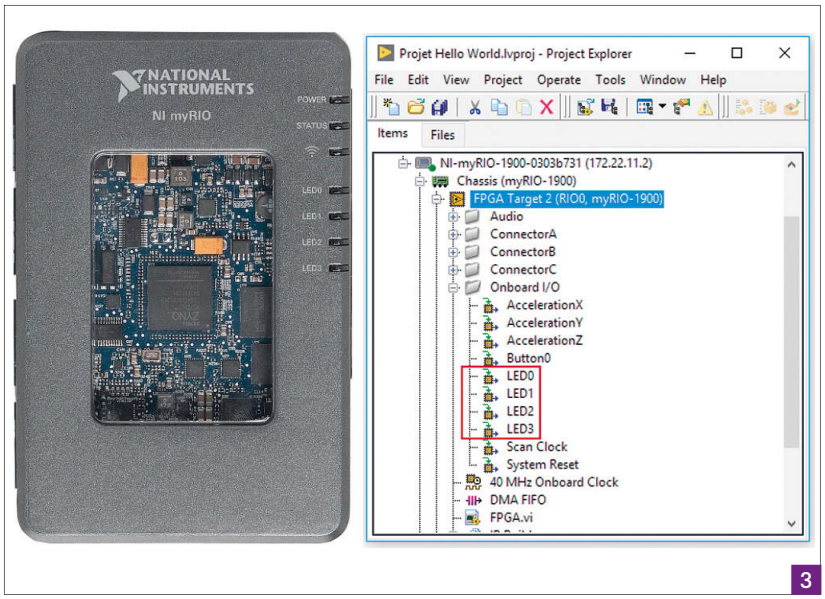
Un code LabVIEW FPGA se compose de trois éléments :

- Un projet, véritable squelette de l'application ;
- Un diagramme comprenant le code source ou la logique FPGA ;
- Une face-avant ou interface d'éléments qui permet de visualiser les entrées et sorties connectées. 2

2



Dès sa création, le projet, via un grand nombre d'exemples disponibles pour différentes cibles matérielles, permet de visualiser l'architecture matérielle utilisée : de la puce FPGA bien sûr, jusqu'aux éléments plus haut niveau comme le processeur temps-réel ou le PC de développement. On crée ensuite un fichier *mon_application.vi* (VI pour instrument virtuel), l'équivalent du code source, qui comprend un diagramme et une face-avant. *mon_application.vi* va alors comprendre la logique propre de l'algorithme à implémenter, comme l'acquisition d'un phénomène vibratoire et son analyse fréquentielle par exemple, ou encore un traitement d'images issues d'une caméra. Très vite, la logique graphique de LabVIEW permet de développer le code en suivant le flux de données théorique mais surtout l'indépendance des tâches : deux fonctions à implémenter sur le FPGA équivalent à deux boucles graphiques dans le code source.

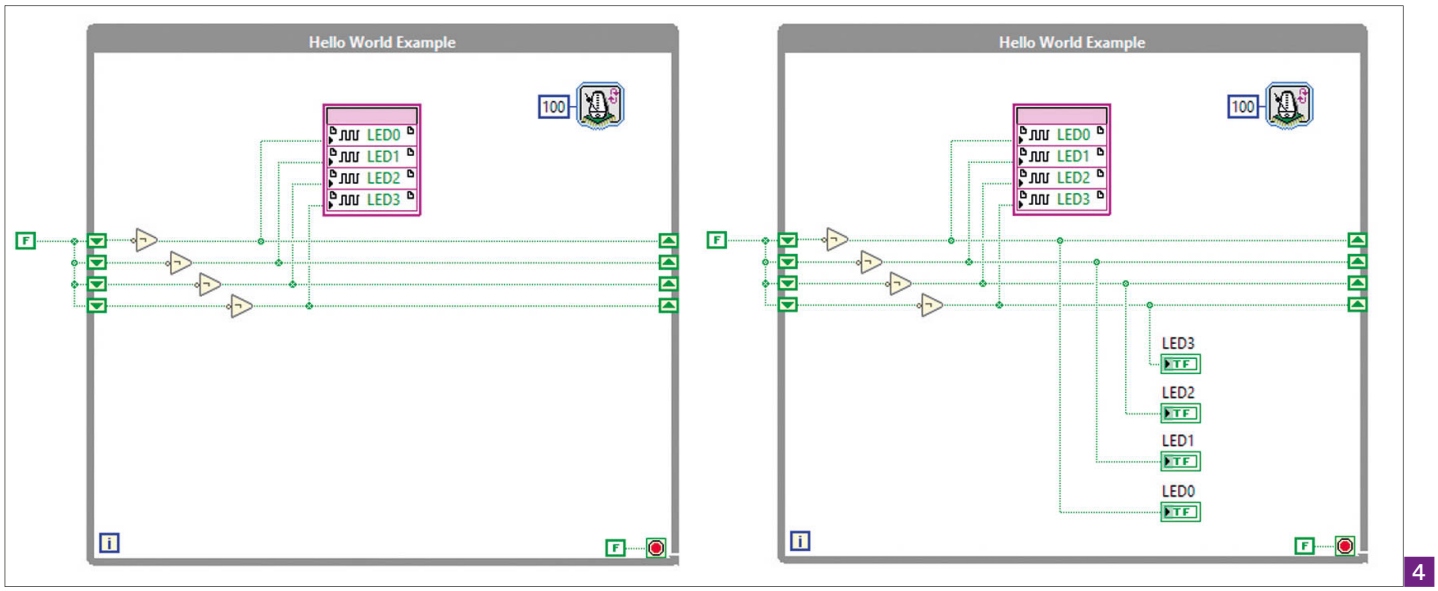


3

Pour illustrer ce processus nous allons créer un *Hello World* « matériel », à savoir le pilotage et clignotement de plusieurs LEDs. Dans cet exemple, nous utiliserons une plateforme matérielle myRIO. Cette cible se compose d'un processeur programmable double-cœur ARM Cortex-A9 et d'un FPGA Xilinx Artix 7. Le matériel myRIO repose sur le système sur puce (SoC) entièrement programmable Zynq-7010. Toutes ces ressources deviennent disponibles une fois la cible matérielle ajoutée au projet (voir figure précédente).

Pour mettre en œuvre ce programme, nous créons un code source utilisant les ressources natives de la cible, à savoir les quatre LEDs disponibles : LED0, LED1, LED2 et LED3 comme on peut le voir sur la figure suivante. 3

Dans cette petite application, nous allons donc créer la boucle de gestion des LEDs. Pour ce faire il est nécessaire de glisser/déposer l'élément graphique correspondant, à savoir une boucle « tant que »



4

(ou boucle *while* en anglais) dans laquelle nous ajouterons les ressources physiques ou entrées/sorties observées précédemment. Nous allons ensuite spécifier le cadencement de cette boucle, c'est-à-dire la vitesse d'exécution du code, dans notre cas un cadencement à 100 ms soit 10 Hz. Pour information, il est possible d'utiliser une boucle cadencée à une seule période pour créer par exemple un compteur 40 MHz sur n'importe quelle ligne numérique. Cette horloge de 40 MHz est disponible sur le FPGA utilisé dans notre exemple.

À présent nous allons ajouter la fonction « non » booléenne permettant de changer l'état des LED à chaque itération de la boucle. Pour cela, nous utilisons la fonction « Non » depuis la palette Boolean et la câblons à l'extrémité gauche de la boucle et au connecteur de LED0. En entrée de la boucle nous allons ajouter une valeur initiale, via la constante.

Enfin, pour pouvoir mettre en mémoire la valeur booléenne d'une itération à une autre, nous allons avoir besoin d'un mécanisme de rétention de données, via un registre à décalage. Nous répétons ensuite cette étape pour les quatre LEDs comme sur la figure ci-dessous. Pour les observer sur notre face-avant, nous ajoutons quatre indicateurs LEDs disponibles dans la palette, puis les câblons. 4

Le code est prêt à être compilé pour ensuite être téléchargé sur la puce FPGA et fonctionner de manière totalement autonome. Cette étape de compilation, expliquée plus haut, est inhérente au développement sur FPGA, et elle peut être optimisée selon plusieurs critères, comme par exemple les ressources utilisées, ou encore le temps global de compilation.

Conclusion

Même si le développement sur FPGA peut paraître fastidieux et moins accessible que celui sur d'autres cibles matérielles, l'utilisation d'environnement graphique et de langages d'abstraction permet de rapidement monter en compétences. Il ne reste plus qu'à les prendre en main, et pour cela des kits de développement sont disponibles pour démarrer sereinement et à prix réduit. Maintenant, à vous de jouer ! •

Initiation à LabVIEW FPGA :
http://www.ni.com/tutorial/14532/fr/
Kits d'évaluation LabVIEW RIO :
http://sine.ni.com/nips/cds/view/p/lang/fr/nid/205721

Bruno MESNET
CAPI SNAP Proof of Concepts Team leader
Bruno est Ingénieur avant-vente chez IBM Systems. Il s'intéresse au portage de toutes les applications qui peuvent être accélérées. Basé à Montpellier, l'expérience client est son moteur à travers le précepte : "le plus simple, pour obtenir le meilleur".

Boostez vos applications grâce aux FPGA.

Enfin une technologie accessible aux développeurs logiciels !

Un exemple concret ? Une fonction de calcul de clef de cryptographie de type SHA3 écrite en C a été portée sur un FPGA en 11 jours pour des résultats montrant un facteur d'accélération de 35 par rapport à l'exécution sur un serveur avec des CPU multi-threadés. Aucune modification du code de l'algorithme lui-même, n'a été nécessaire. Aucune connaissance spécifique aux FPGA ou à ses outils ne fût nécessaire. Convaincant ?

Cela vous semble un peu trop « magique » pour être vrai ? Mais qu'y a-t-il de si nouveau qui pourrait révolutionner le monde de l'IT ? On connaît déjà les GPU, mais qu'est-ce qu'un FPGA a de si différent ? Et pourquoi ce FPGA, qui est une vieille technique, pourrait redevenir subitement intéressant ?
Rappelons ce qu'est un FPGA, pour comprendre les raisons qui ont limité le développement de cette technologie dans l'IT (langage spécifique, driver, …). "OpenPower", le consortium créé autour de la technologie du processeur Power d'IBM a donc mis en place un environnement de développement que nous vous présentons et qui permet de vulgariser l'utilisation de cette technologie.

Rappels

Le FPGA, le composant programmé pour VOTRE fonction. Ces 4 lettres FPGA qui veulent dire « Field Programmable Gate Array » désignent un **composant reprogrammable** qui contient de la logique combinatoire (portes logiques OR, AND, XOR, additionneurs, multiplieurs) mais aussi de la mémoire et des entrées/sorties. Ces éléments répétés un nombre immense de fois, sont répartis à l'aide d'une matrice d'interconnexion qui permet de relier toutes ces ressources entre elles. Grâce à cela, on va pouvoir créer « sur mesure » des fonctions logiques parfaitement optimisées pour LA fonction que vous voulez externaliser, voire accélérer. Ce concept est très important à comprendre car contrairement à un processeur (CPU ou GPU), on n'adapte pas la « fonction à exécuter » à la logique mais on **construit la logique pour la fonction à exécuter**. Par extension, on comprend facilement, qu'en fonction de la taille prise par la logique, on peut facilement la dupliquer pour paralléliser le traitement autant de fois que le FPGA peut contenir de duplications ! Une fois la logique construite, le FPGA est programmé et l'utilisateur pourra exécuter la fonction programmée jusqu'à ce que l'on décide de reprogrammer ce FPGA.

Le choix du FPGA

Ces FPGA créés en 1985 contiennent de plus en plus de logique et de mémoire au fur et à mesure des nouvelles générations, mais contiennent aujourd'hui aussi des **ressources très spécifiques** comme des processeurs ARM, des DSP, ou des liens très haute vitesse. Les 2 principaux fabricants sont « Intel FPGA », anciennement « Altera », et « Xilinx », mais la diversité provient surtout des fabricants de cartes utilisant ces FPGA. On trouvera ainsi un nombre quasi illimité de cartes adaptées à tous les standards possibles en fonction de l'utilisation que l'on veut en faire. Les cartes FPGA que l'on utilise dans les serveurs seront par exemple connectées à un slot PCIe, et contiennent soit beaucoup de mémoire (2TB de Flash), soit des connecteurs permettant de s'interfacer avec le monde extérieur : réseau, disques, etc. Le choix du FPGA et de la carte associée, sont donc essentiellement déterminés par l'utilisation que l'on veut en faire.

Programmer un FPGA en C/C++, ... enfin.

Savoir ce qu'est un FPGA et comment le choisir est une chose. Le **programmer** en est une autre… et c'est bien là le point de blocage du FPGA depuis des années ! Jusqu'à aujourd'hui, seuls des développeurs dits « hardware » étaient en mesure de coder ces composants. Les langages HDL (Hardware Description Language) de très bas niveaux comme le Verilog ou le VHDL décrivent la logique élémentaire. Toute la logique est ensuite connectée, puis cadencée avec une horloge, et nécessite des connaissances extrêmement spécifiques et différentes des connaissances logicielles. Beaucoup de sociétés se sont essayées à faire des « compilateurs » ou ont créé des nouveaux langages (SystemC, OpenCL, …) pour permettre à des codeurs d'applications d'utiliser des FPGA de façon efficace. Depuis ces deux dernières années, on commence enfin à voir des **« convertisseurs de code »** C/C++ en langage HDL. Ces outils appelés **HLS (High Level Synthesis)** sont beaucoup plus que des simples traducteurs de langages, car ils doivent **analyser du code essentiellement séquentiel** en comprenant les dépendances entre les entrées et sorties des différentes fonctions pour pouvoir paralléliser l'algorithme. Cette étape effectuée, il faut attribuer et **connecter la logique** pour que cela prenne le moins de place possible dans le composant. Il faut ensuite **cadencer la logique** avec une horloge et vérifier que cette logique est synchrone sous peine de mauvais calculs ! Ces outils HLS sont principalement délivrés par les fabricants de FPGA (Intel, Xilinx) ou d'outils de design de composants (Cadence, Mentor Graphics).

Améliorer l'ensemble de la chaîne "stockage - réseau - CPU".

Il est intéressant de noter 2 points très importants à ce stade :

- Le **cadencement d'un processeur** se fait à 3 ou 4 GHz, alors qu'un GPU sera cadencé à environ 1 GHz et un FPGA à … 250MHz. Le fait que l'on puisse créer la logique « à façon » et la dupliquer va permettre de gagner de la performance malgré cette fréquence moindre. N multiplications sur un CPU nécessitent N cycles d'horloge alors que le FPGA pourra les effectuer en 1 seul cycle d'horloge !

- Le FPGA permet de stocker les données dans la mémoire de la carte, ou de les récupérer directement sur ses connecteurs externes. Cela permet de ne pas utiliser la carte réseau du serveur et donc de gagner de la latence, mais surtout de la bande passante sur les cartes réseaux. Intégrer un FPGA ne se limite pas à « sous-traiter » une tâche, mais permet d'**améliorer l'ensemble de la chaîne « stockage externe – réseau – stockage mémoire– CPU ».**

Un point important à noter aussi est que le FPGA apporte toujours un bien meilleur rapport puissance électrique consommée / performance d'exécution d'une fonction, que les GPU ou même les CPU.

Le FPGA, plus de performances mais encore trop de contraintes ?

En résumé, une carte contenant un FPGA permet :

- **D'externaliser** un programme habituellement exécuté sur le CPU, et donc de réduire l'utilisation du temps CPU,
- **D'accélérer** la fonction externalisée par une logique optimisée et parallélisée,
- De profiter de **ressources nouvelles** disponibles sur la carte, donc de réduire l'utilisation des ressources du serveur
- De **diminuer la consommation électrique** en optimisant l'ensemble de la chaîne d'exécution.
- D'être programmé avec des langages évolués tel que le C/C++

Jusqu'à présent, l'utilisation d'une carte externe apportait, en général, aussi des contraintes à ne pas sous-estimer :

- Gestion des **mouvements de données** de/vers la carte externe à **tâche supplémentaire** à coder dans l'application appelante,
- Gestion des **nouvelles ressources** apportées par la carte FPGA et inconnues du serveur à **tâche supplémentaire** à coder dans la fonction implémentée dans le FPGA ,
- Utilisation d'un **driver** de la carte qui **requiert du temps CPU** mais aussi des ressources **mémoires** du serveur,
- **Dépendance de la fonction « hardware »** vis-à-vis de l'application appelante.

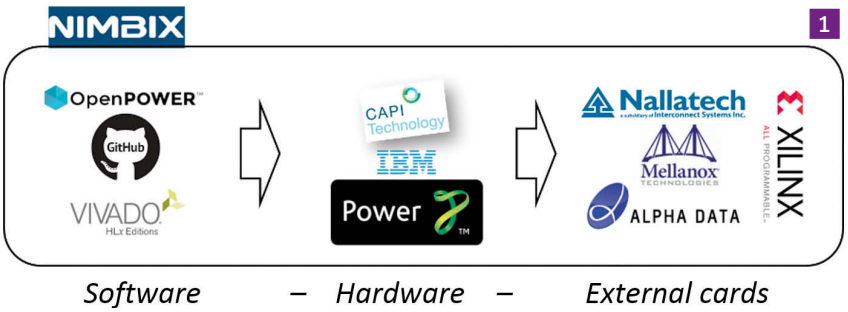
Et si le FPGA etait autonome ?

IBM, avec ses partenaires, a donc travaillé sur ces 4 points essentiels pour qu'un **développeur logiciel** puisse :

- Donner au FPGA des **droits identiques aux processeurs** : accès direct et cohérent à la mémoire du serveur, autonomie, …
- **Libérer complètement son CPU** et la mémoire anciennement associée au driver dès lors qu'une activité est sous-traitée,
- Accélérer son code **sans apprendre un nouveau langage**,
- Porter son code en le **modifiant le moins possible**.

La solution développée est alors composée de 3 éléments distincts :

- Une **interface** intégrée dans la puce du processeur qui donne à une carte extérieure les mêmes droits d'accès à la mémoire qu'un des processeurs du serveur, sans la nécessité d'insérer un driver logiciel. Cette interface nommée **CAPI** (Coherent Accelerator Processor Interface) a été développée dans le cadre de la fondation OpenPower pour la première fois pour le processeur **IBM Power8,** et bien sûr, grandement améliorée pour le processeur **Power9** (disponible depuis fin 2017),
- Un « **framework** » qui permet de donner à un programmeur logiciel un environnement de développement simple pour qu'il puisse porter son application sur une carte dont la logique est faite sur mesure pour accélérer son algorithme. Ce framework **open-source** nommé **SNAP** (Storage, Networking, Analytics Programming) est développé sur des cartes à base de FPGA,
- Une **carte FPGA** comprenant des ressources différentes suivant les besoins de la fonction à accélérer (2TB de Flash, 4GB de DDR4, 8GB de DDR3, 2 ports 40Gb QSFP+, …).



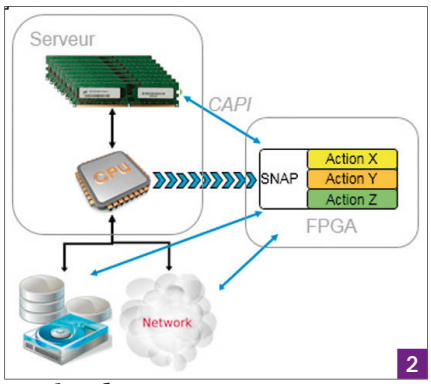
1

Un développement en partenariat

Ce framework développé en collaboration entre IBM et ses partenaires OpenPower, a permis d'assembler le meilleur de la technologie pour parvenir à proposer cette solution innovante et performante dans un environnement ouvert. 1

Cette solution simple et performante est enfin proposée à des développeurs d'applications grâce à l'intégration :

- de l'interface CAPI du processeur POWER ;
- d'un **FPGA**
- de fonctions qui peuvent être programmées en C/C++ grâce à **Vivado HLS**, seul outil HLS utilisable sans connaissances « hardware »,
- accès aux différentes ressources simplement grâce au framework **SNAP** contenant des **API** qui simplifient l'interfaçage avec cette carte extérieure. 2



2

Un seul de ces 4 éléments manque et la « magie » n'opère plus !

Concrètement, lorsqu'une fonction d'une application est repérée et identifiée comme prenant une partie importante du temps CPU, il est judicieux de voir comment l'optimiser. Il peut s'agir de l'accélérer mais peut être juste dans un premier temps de l'externaliser. Sachant que le FPGA a un accès direct à la mémoire du serveur, il suffit de donner à la fonction qui est dans le FPGA l'adresse où se trouvent les données et il ira les chercher tout seul sans l'aide de l'application ni d'un driver.

Le portage d'une fonction à accélérer:

Ce portage va se faire en 4 phases :

- **Séparation** de « la fonction à externaliser » de « l'application appelante » et **intégration** de l'application, et de cette fonction « software » dans le framework SNAP ;
- **Adaptation** de la « fonction à externaliser » « software » en « hardware » et **simulation** dans l'environnement SNAP avec un modèle de l'interface CAPI du processeur POWER ;
- **Exécution** de la « fonction à externaliser » « hardware » et **mesure** de la performance sur la carte FPGA dans un serveur IBM Power Systems ;

• **Optimisation** du code pour obtenir les **meilleures performances**.

Précisons que la fonction « software » est exécutée sur le CPU, par opposition à la fonction « hardware » qui est exécutée sur le FPGA. On garde intentionnellement ces 2 fonctions en parallèle pour mesurer des performances comparatives mais aussi pour permettre de localiser et debugger plus facilement les erreurs que l'on pourrait rencontrer.

Une fois le framework SNAP installé (https://github.com/open-power/snap), la fonction à externaliser en C/C++ est donc d'abord **compilée** avec l'outil « Vivado HLS » de Xilinx qui va générer du code HDL décrivant la logique compréhensible par le FPGA. Des contraintes, dues principalement à l'absence de système d'exploitation sur le FPGA et l'accès aux données en mémoire, vont exiger quelques adaptations (appels systèmes à proscrire, allocations dynamiques de mémoire à transformer en allocation fixes, etc.).

Une fois le code de la fonction compilé, il s'agit d'**adapter l'application** pour que celle-ci puisse s'interfacer avec la fonction. En d'autres termes, l'application va s'attacher à une fonction se trouvant dans une carte FPGA. Le framework SNAP sait effectivement gérer plusieurs cartes par serveur mais aussi plusieurs « actions » par cartes. SNAP ayant la liste de toutes les actions disponibles dans le serveur, saura donc affecter ou mettre en file d'attente une demande faite par une application pour accéder à une action. Le code de l'application devra donc contenir l'assignation de la carte et de la fonction comme suit :

```
card = snap_card_alloc_dev(device, SNAP_VENDOR_ID_IBM, SNAP_DEVICE_ID_SNAP);
action = snap_attach_action(card, HELLOWORLD_ACTION_TYPE,
                (SNAP_ACTION_DONE_IRQ | SNAP_ATTACH_IRQ), timeout);
.../...
snap_detach_action(action);
snap_card_free(card);
```

L'interfaçage entre l'application appelante et la fonction se fait via 2 canaux :

- une structure de registres pour échanger un nombre limité des paramètres (MMIO) ;
- la zone mémoire du serveur pour échanger un grand nombre de données. Seule l'adresse de cette zone sera donnée dans les registres.

Des API pour une gestion sans peine du FPGA

Le nombre très restreint d'API permet donc d'échanger comme ci-dessous :

```
/* pour lire et écrire un registre du FPGA via des structures d'entrée et de sortie*/
mjob_in->chk_in = chk_in_value;
mjob_out->chk_out = 0x0;
snap_job_set(cjob, mjob_in, sizeof(*mjob_in), mjob_out, sizeof(*mjob_out));
/* pour passer l'adresse de la zone mémoire du serveur ou se trouvent les données */
snap_addr_set(&mjob->out, addr_out, size_out, type_out,
        SNAP_ADDRFLAG_ADDR | SNAP_ADDRFLAG_DST |SNAP_ADDRFLAG_END);
/* pour donner l'ordre de démarrage à la fonction */
rc = snap_action_sync_execute_job(action, &cjob, timeout);
```

Souvenons-nous que la fonction dans le FPGA est totalement autonome. Le paradigme de passer des arguments à une fonction et de récupérer le résultat change donc ici. L'application configure la fonction et lui dit de démarrer. Les résultats seront disponibles directement en mémoire lorsque l'application en aura besoin. Elle peut être informée par une interruption que le travail est terminé, ou vérifier à tout moment où en est l'exécution.

Une fois ce code compilé, il va pouvoir être **simulé** pour vérifier que ces modifications n'ont pas altéré la fonctionnalité du code. Le test de ce code se présente alors comme le test d'un code C classique avec des « printf », mais on peut aussi vérifier très simplement les échanges entre l'application et la fonction en rajoutant une option avant l'appel de l'application.

```
SNAP_TRACE=0xF  snap_helloworld –i t1 –o t2
.../...
R  hw_snap_mmio_write32(0x22c4010, f100, f0020100)
R  hw_snap_mmio_write32(0x22c4010, f104, 0)
R  hw_snap_mmio_write32(0x22c4010, f108, c0febabe)
R  hw_snap_mmio_write32(0x22c4010, f10c, deadbeef)
R  hw_snap_mmio_read32(0x22c4010, f000, 6) 0
.../...
```

Le code final, qui est le programme binaire de la fonction exécutée dans le FPGA, pourra alors être généré et exécuté.

Aujourd'hui, (en plus des langages HDL classiques,) le langage supporté par le framework SNAP est le **C/C++** grâce à l'outil « Vivado HLS » de Xilinx. Certains de nos partenaires travaillent pour supporter d'autres langages comme le langage **Go**, mais gardons en mémoire que le C/C++ est le standard qui aujourd'hui permet de disposer d'une multitude de passerelles de/vers d'autres langages tels que le **Java** avec JNI.

L'essayer, c'est l'adopter

Pour permettre à tout développeur d'essayer cette technologie, et de mesurer par lui-même la simplicité et l'efficacité de leur portage, notre partenaire Nimbix (https://www.nimbix.net/) met à disposition (*disponible en Mars 2018*) un environnement complet de développement (serveurs, licences, cartes FPGA). Cela permet, moyennant un coût de connexion de 0,36$ par heure, de développer, découvrir, évaluer et apprécier cette technologie, le test final sur FPGA quant à lui est à 3$ par heure. Des exemples variés et complets sont donnés pour ne pas partir d'une page blanche. Ils permettent de découvrir la façon de coder.

Commencez par découvrir notre exemple **hello_world** (https://github.com/open-power/snap/tree/master/actions/hls_helloworld) qui permet de changer la casse d'un texte à travers seulement 4 fichiers sources.

L'appel des fonctions « software » et « hardware » se fait grâce à un flag ajouté au moment de l'appel de **snap_helloworld** qui est le fichier exécutable de la fonction C appelante avec comme arguments d'entrée les fichiers de texte t1 à convertir en t2 :

```
SNAP_CONFIG=CPU snap_helloworld –i t1 –o t2
(SNAP_CONFIG=FPGA) snap_helloworld –i t1 –o t2
```

Ainsi, en juste un peu plus d'**une heure,** sans aucune connaissance, ni matériel, ni FPGA, vous pourrez installer le framework, sélectionner un exemple, regarder, voire modifier le code, le compiler, le simuler et construire le fichier binaire pour le FPGA pour finalement l'exécuter sur un vrai FPGA. •

Plus d'informations sur https://developer.ibm.com/linuxonpower/capi/snap/ *ou* ***ibm.biz/powercapi_snap***

Pour toute question ou information, n'hésitez pas à écrire à : capi@us.ibm.com

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

La Programmation Orientée Objet en C++

Il y a plusieurs façons d'utiliser le C++. On peut être utilisateur de classes ou concepteur de classes. Dans les deux cas, définir, concevoir et implémenter des classes sont les activités primaires des développeurs C++.

Par où on commence ? En général, on démarre avec une abstraction ou un concept. On considère un truc à coder et puis on essaie de voir comment on va gérer son ou ses fonctions, ce qui est interne et ce qui est public et comment on va l'utiliser… Au démarrage de cette réflexion, il faut se mettre la casquette de celui qui va utiliser la classe. Notre exemple de concept pour cet article sera un élément graphique à dessiner comme une ligne, un rectangle, une ellipse, etc. En anglais, c'est un shape.

```
// C'est une déclaration de classe
class CShape;

// C'est la future manière avec laquelle je veux dessiner des Shapes
bool Draw(const CShape& item);
```

Ensuite on envisage la classe dans son ensemble :

```
class CShape
{
public:
  // interface publique

private:
  // implémentation privée
};
```

Qu'est-ce que l'on met en private/public et comment va-t-elle être structurée. On commence petit bras et puis la classe prend du volume.

Constructeur et Destructeur

La première question qui vient à l'esprit est comment vais-je créer mon objet et ai-je besoin de lui fournir des paramètres ? Avec ma classe CShape, on peut se dire, je n'en ai pas besoin mais aussi j'en ai besoin. Les deux possibilités existent ! Soit c'est une figure connue -> d'où une énumération pour les types connus, et, autre possibilité qui consiste à dessiner une image, et là, il me faut un nom de fichier ; tout est ouvert !

```
enum ShapeType
{
  line,
  circle,
  rectangle,
  left_arrow,
  right_arrow,
  picture
};
```

L'avantage de l'énumération est que c'est évalué à la compilation. Il ne peut pas y avoir d'erreur sur le nommage car c'est un type explicite contrairement à une simple chaîne de caractères. Voyons sommairement comment pourrait être construite cette fameuse classe CShape…

```
class CShape
{
public:
  CShape();
  CShape(int x, int y, int xx, int yy, ShapeType type);
  CShape(int x, int y, int xx, int yy, std::string fileName);
  virtual ~CShape();

private:
  std::string m_fileName;
  ShapeType m_type;
};
```

J'y vois deux façons de créer un shape. Soit à partir de l'énumération soit à partir d'un fichier pour les images. Pour le moment, je ne sais pas comment organiser le dessin du shape. Dois-je le mettre à l'intérieur de la classe ou à l'extérieur ? Je ne sais pas… Il faut du temps pour considérer quelle sera la façon la plus naturelle de réaliser cette opération. Dans le monde Windows GDI+, pour dessiner un élément il faut un handle particulier qui possède toutes les primitives de dessins. En pensant comment allait être dessiné un élément, on peut envisager d'inclure une méthode Draw dans la classe CShape. La première pensée, c'est de se dire, je vais implémenter tous les cas possibles dans Draw avec une construction qui ressemble à ça :

```
void Draw(const CDrawingContext& ctxt) const
{
    if (m_type == ShapeType::line)
    {
        ctxt.Line(m_rect.left, m_rect.bottom,
            m_rect.top, m_rect.right);
    }
    if (m_type == ShapeType::rectangle)
    {
        ctxt.Rectangle(m_rect);
    }
    // TODO
}
```

Dans la classe, il est important de marquer les méthodes qui ne modifient pas les données private. Pour faire cela, on leur ajoute le

mot-clé const à la fin comme c'est indiqué sur la méthode Draw.
De plus, lorsque l'on passe un argument qui n'a pas vocation à être modifié, on le passe en référence const. Comme on peut le voir ci-dessus, la définition d'une classe est un travail itératif dans lequel les meilleures idées chassent celles d'avant ou les conforte. Si une classe est bien conçue, la lecture de ses membres public est limpide car on va à l'essentiel. Attention, les membres private expliquent aussi beaucoup de choses sur les détails d'implémentation. Maintenant, mettons-nous dans un contexte où il existe plusieurs classes et les principes vus ci-dessus ne suffisent pas à faire un modèle objet. Il nous faut des mécanismes plus sophistiqués pour relier les classes les unes avec les autres.
Les classes possèdent des associations et des comportements, et c'est ici que les Jedi se positionnent pour y concevoir un modèle élégant, facile à utiliser et qui rend les services voulus.
Notre méthode Draw n'est pas OOP car si je rajoute un type dans l'énumération, je dois ajouter un if() dans la méthode Draw() et coder l'implémentation. On peut faire mieux, beaucoup mieux.

Les concepts de OOP

Les deux piliers de l'OOP sont l'héritage et le polymorphisme. L'héritage permet de grouper les classes en familles de types et permet de partager des opérations et des comportements et des données. Le polymorphisme permet de déclencher des opérations dans ces familles à un niveau unitaire. Ainsi ajouter ou supprimer une classe n'est pas trop un problème.
L'héritage définit une relation parent/enfant. Le parent définit l'interface publique et l'implémentation private est commune à tous ses enfants. Chaque enfant choisit s'il veut hériter d'un comportement unique ou bien le surcharger. En C++, le parent est appelé « classe de base » et l'enfant « classe dérivée ». Le parent et les enfants définissent une hiérarchie de classes. Le parent est souvent une classe abstraite et les enfants implémentent les méthodes virtuelles pures pour que l'ensemble fonctionne.
Dans un programme OOP, on manipule les classes via un pointeur sur la classe de base plutôt que sur les objets dérivés.

Une classe abstraite

Revenons sur la classe CShape… La prochaine étape dans le design est de créer une classe abstraite pour identifier les opérations propres à chaque item – ce qui implique des opérations avec une implémentation basée sur une classe dérivée. Ces opérations sont des fonctions virtuelles pures de la classe de base. Une méthode virtuelle pure est une méthode abstraite. Il n'y a pas de corps. Cela implique que la classe devient abstraite aussi. Il n'est pas possible de créer un objet à partir d'une classe abstraite. La méthode Draw() doit être définie comme virtuelle pure car il y a plusieurs types de dessin à faire et que chaque dessin est propre à une classe donnée. Le corps de Draw() n'existe pas, c'est la raison pour laquelle on dit que la classe est abstraite ; il y a au moins une méthode virtuelle pure. Voici donc comment on fait :

```
class CShapeEx
{
public:
  CShapeEx() {}
  CShapeEx(ShapeType type, RECT rect, const std::string &fileName)
      : m_type(type), m_rect(rect), m_fileName(fileName) {}
  virtual ~CShapeEx() {}

public:
  virtual void Draw(const CDrawingContext& ctxt) const = 0;
  virtual void DrawTracker(const CDrawingContext& ctxt) const = 0;

private:
  std::string m_fileName;
  ShapeType m_type;
  RECT m_rect;
};
```

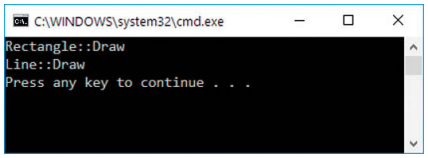
Et nous allons maintenant déclarer des classes dérivées de CShapeEx :

```
class CRectangle : public CShapeEx
{
public:
  CRectangle() {}
  virtual ~CRectangle() {}

public:
  virtual void Draw(const CDrawingContext& ctxt) const
  {
    // TODO
    std::cout << "Rectangle::Draw" << std::endl;
  }

  virtual void DrawTracker(const CDrawingContext& ctxt) const
  {
    // TODO
  }
};

class CLine : public CShapeEx
{
public:
  CLine() {}
  virtual ~CLine() {}

public:
  virtual void Draw(const CDrawingContext& ctxt) const
  {
    // TODO
    std::cout << "Line    ::Draw" << std::endl;
  }

  virtual void DrawTracker(const CDrawingContext& ctxt) const
  {
    // TODO
  }
};
```

Si on veut rajouter un item à dessiner dans la hiérarchie, il suffit de rajouter une classe dérivée ! Maintenant nous allons voir comment fonctionne ce mécanisme de virtual…

```
CDrawingContext ctxt;
CRectangle * pRect = new CRectangle();
CLine * pLine = new CLine();
CShapeEx * ptr = nullptr;
ptr = pRect;
ptr->Draw(ctxt);
ptr = pLine;
ptr->Draw(ctxt);
```

On commence par déclarer 2 items à dessiner. Puis le manager, un pointeur sur la classe de base CShapeEx est déclaré. A chaque fois qu'il pointe sur un objet d'une classe dérivée, la fonction virtuelle Draw() appelée est celle de la classe dérivée car cette fonction est définie comme virtuelle.



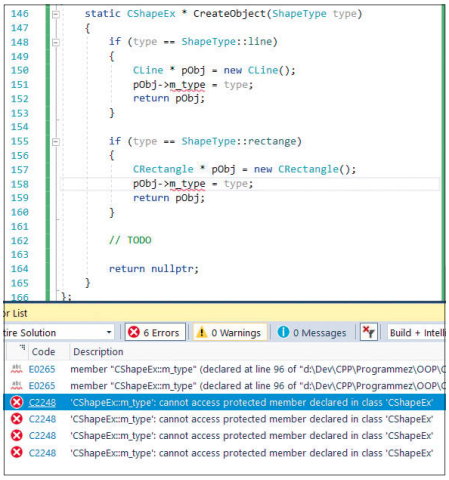
Maintenant, il nous manque quelque chose de très utile qui évitera les usines à gaz de construction d'objets dérivés. Il nous faut une factory ! Un mécanisme de création d'objet.

La factory

```
class CFactory
{
private:
  CFactory();

public:
  static CShapeEx * CreateObject(ShapeType type)
  {
    if (type == ShapeType::line)
    {
      CLine * pObj = new CLine();
      pObj->m_type = type;
      return pObj;
    }

    if (type == ShapeType::rectange)
    {
      CRectangle * pObj = new CRectangle();
      pObj->m_type = type;
      return pObj;
    }

    // TODO

    return nullptr;
  }
};
```

Cette classe permet via une méthode static de créer un objet en fonction de son type dans l'énumération. Le type est affecté en variable membre de l'objet créé. On notera que le constructeur est private, cela veut dire que l'on ne peut pas déclarer d'objet CFactory.

On ne peut qu'utiliser la méthode static CreateObject. Sauf que, la compilation tombe en erreur :



Le membre m_type est inaccessible vu son niveau de protection… On va donc ajouter quelque chose à la classe CShapeEx :

```
class CShapeEx
{
public:
  CShapeEx() {}
  CShapeEx(ShapeType type, RECT rect, const std::string &fileName)
    : m_type(type), m_rect(rect), m_fileName(fileName) {}
  virtual ~CShapeEx() {}

public:
  virtual void Draw(const CDrawingContext& ctxt) const = 0;
  virtual void DrawTracker(const CDrawingContext& ctxt) const = 0;

private:
  std::string m_fileName;
  ShapeType m_type;
  RECT m_rect;

  // La classe CFactory peux avoir accès aux membres...
  friend class CFactory;
};
```

On ajoute en fin de classe que la classe CFactory est amie de la classe CShapeEx. Et là, il n'y a plus d'erreur de compilation ! Les puristes comme AlainZ, à Redmond, vous diront que cela viole la mécanique objet et que c'est une construction exotique et qu'il vaut mieux passer par le constructeur pour passer le type… Je ne suis pas de cet avis. Je pense que c'est élégant de faire une entorse pour la factory. Voici maintenant le code qui appelle la factory :

```
CDrawingContext ctxt;
CShapeEx * pRect = CFactory::CreateObject(ShapeType::rectange);
CShapeEx * pLine = CFactory::CreateObject(ShapeType::line);
CShapeEx * ptr = nullptr;
ptr = pRect;
ptr->Draw(ctxt);
ptr = pLine;
ptr->Draw(ctxt);
```

Et le résultat est équivalent.

Compilation sous Linux

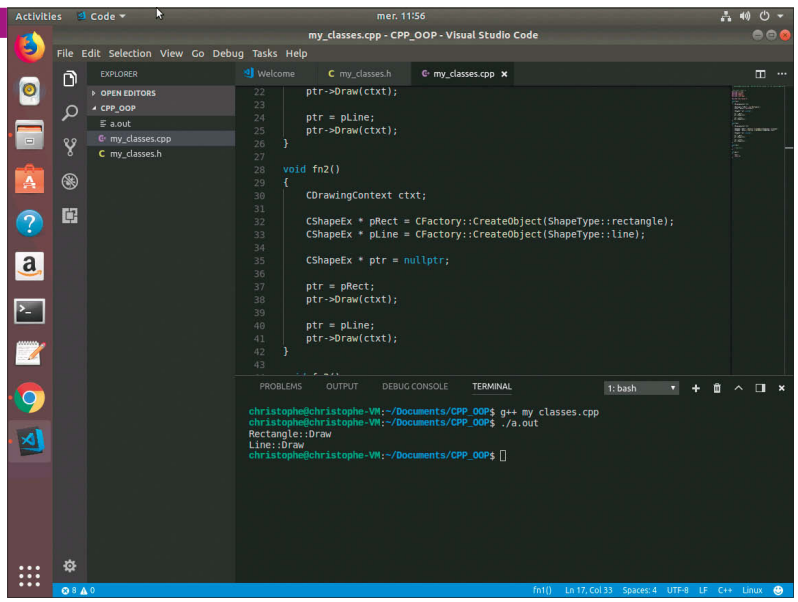
Pour ceux qui ne le savent pas, le langage C++ est normalisé ISO. Sur la plateforme Linux, là où tout est fait en C/C++, le langage possède plusieurs compilateurs dont le célèbre GCC. Vous pouvez utiliser Visual Studio Code que fournit Microsoft comme IDE pour compiler sous Linux via le terminal.

Il faut installer l'extension C++. Moi j'ai installé Cygwin et MSYS. Ainsi je dispose de commandes Linux. 1

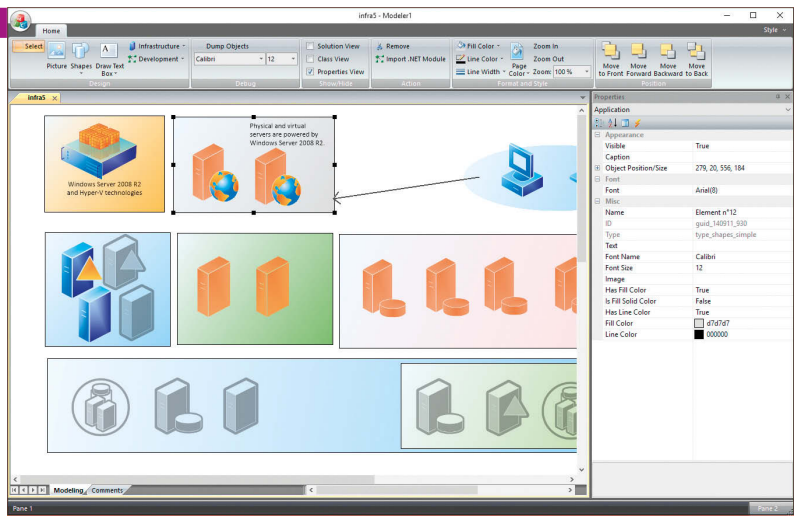
Une touche de C++ 11

On va essayer de continuer notre application de dessin en matérialisant la zone de dessin. Cela ressemble à un ensemble de shapes sur un support de dessin. Première étape quand on pense au concept de la planche à dessin c'est quoi ? Le stockage des éléments graphiques dans un container STL. Oui Monsieur, quand on veut stocker quelque chose, on passe par les containers STL. On va donc utiliser le std::vector<T> de la STL. On va stocker des shapes un peu particulières que sont les shapes partagées. Partagées ? Par qui ? Dans un vrai logiciel de dessin, on affiche les propriétés dans une grille et le dessin est sur le panneau central, donc on partage nos objets et ce n'est pas que de l'affichage. Je vous montre le résultat. 2

1



2



Sur cet écran, on voit que le rectangle gris sélectionné a ses attributs directement dans la grille de propriétés mais aussi sur le Ribbon. Comment faire pour partager les données entre le panneau de dessin, le Ribbon et la grille de propriétés ? Est-ce que l'on fait des copies des objets ? Oui, mais on fait des copies de pointeurs et non des copies d'objets. Pour faire cela, on utilise le mécanisme des pointeurs partagés de la STL. Voyons son utilisation. Premièrement, on va concevoir graphiquement la zone de dessin.

La zone de dessin

Cette zone contient les objets shape à dessiner. On va donc créer un vector<shared_ptr<CShapeEx>>. Le container vector est défini dans <vector>. On va avoir une collection d'éléments partagés CShapeEx. Ainsi, on pourra stocker ce pointeur particulier où on veut sans se soucier de la libération de la mémoire. La zone de dessin est une vue fille MDI en jargon Windows. Elle possède deux scrollbars, une verticale et une horizontale. Si on veut faire une gestion de shared_ptr<T>, il faut faire évoluer la factory :

```
static std::shared_ptr<CShapeEx> CreateObject(ShapeType type)
{
    std::shared_ptr<CShapeEx> pObj = nullptr;

    if (type == ShapeType::line)
    {
        pObj = std::make_shared<CLine>();
        pObj->m_type = type;
        return pObj;
    }

    if (type == ShapeType::rectangle)
    {
        pObj = std::make_shared<CRectangle>();
        pObj->m_type = type;
        return pObj;
    }

    return nullptr;
}
```

On ne fait plus un new() mais un appel à std::make_shared<T>(). De plus, dans l'utilisation des objets, cela donne ça :

```
CDrawingContext ctxt;
std::shared_ptr<CShapeEx> pRect = CFactory::CreateObject(ShapeType::rectangle);
std::shared_ptr<CShapeEx> pLine = CFactory::CreateObject(ShapeType::line);
std::shared_ptr<CShapeEx> ptr = nullptr;
ptr = pRect;
ptr->Draw(ctxt);
ptr = pLine;
ptr->Draw(ctxt);
```

Il existe même une façon de laisser le compilateur déterminer le type de variable utilisée en utilisant auto :

```
CDrawingContext ctxt;
auto pRect = CFactory::CreateObject(ShapeType::rectangle);
auto pLine = CFactory::CreateObject(ShapeType::line);
auto ptr = pRect;
```

```
ptr->Draw(ctxt);
ptr = pLine;
ptr->Draw(ctxt);
```

Revenons à notre design de classes. Le container de shapes est juste une collection std::vector<T> de std::shared_ptr<CShapeEx> :

```
class CElementContainer : public CObject
{
private:

public:
  DECLARE_SERIAL(CElementContainer);
  CElementContainer();
  virtual ~CElementContainer(void);

// ../..
// overridden for document i/o
virtual void Serialize(CElementManager * pElementManager, CArchive& ar);
// Debuging Operations
public:
  void DebugDumpObjects(CModeler1View * pView);

// Operations MFC like
public:
  std::shared_ptr<CElement> GetHead();
  int GetCount();
  void RemoveAll();
  void Remove(std::shared_ptr<CElement> pElement);
  void AddTail(std::shared_ptr<CElement> pElement);

// Attributes
private:
  vector<std::shared_ptr<CElement>> m_objects;
};
```

Là, je vous montre un vrai code et les éléments sont matérialisés par la classe CElement mais c'est pareil que CShapeEx. On notera que les fonctions de gestion au vector sont des méthodes helpers dans cette classe. On notera aussi que la fonction Serialize qui permet de faire Load/Save est gérée dans cette classe. En effet, si il existe des donnés à mettre dans un fichier de données, c'est bien cette classe. Il faut maintenant intégrer cette classe dans la mécanique de fenêtrage Windows. Dans un canevas MFC, on possède un Document pour y stocker les données et une View pour y faire les opérations d'affichage.

Abstraction

La couche de dessin est confiée à une classe explicite et les méthodes ne sont pas directement insérées dans la vue. Pourquoi un tel niveau de poupées russes ? Parce que sinon, c'est crado. C'est trop entrelacé dans les couches Windows et le code est spaghetti. J'ai donc une classe qui se nomme CElementManager qui fait le lien avec la View MFC. De ce fait beaucoup de méthodes prennent en paramètre une View… C'est de la délégation de comportement. Mais dans cette classe on trouve tous les éléments de la zone de dessin :

- Les objets à afficher ;
- Les éléments d'une sélection ;
- Les éléments du presse-papier ;
- La couleur de fond de la zone ;
- Le mode du pointeur de souris ;
- Le facteur de zoom ;
- Un booléen pour savoir si on est en train de dessiner un gabarit ;
- L'élément courant avec son type et son id et son nom.

Voici un aperçu du fichier d'entêtes de la classe CElementManager : 3

Cette classe est juste dépendante d'une View sur laquelle il va y avoir des opérations et des manipulations d'affichage. Exemple de méthode :

```
void CElementManager::ZoomIn(CModeler1View * pView)
{
  m_fZoomFactor += 0.10f;
  Invalidate(pView);
}
```

Conclusion

Pour un code orienté objet, il ne faut pas hésiter à faire des constructions, créer des classes et leur donner un minimum de responsabilités. De toute façon, si vous créez une classe et que vous n'y trouvez aucune méthode à mettre dedans, vous devrez faire machine arrière… La construction orientée objet est importante dans le développement moderne et on retrouvera les mêmes principes dans les autres langages comme Java ou C#.

Le code de l'application graphique est disponible ici :
https://github.com/ChristophePichaud/UltraFluidModeler

3

```
// Managing UI dependencies (Ribbon UI, Property Grid, ClassView)
public:
    void UpdateUI(CModeler1View * pView, std::shared_ptr<CElement> pElement);
    void UpdateRibbonUI(CModeler1View * pView, std::shared_ptr<CElement> pElement);
    void UpdatePropertyGrid(CModeler1View * pView, std::shared_ptr<CElement> pElement);
    void UpdateClassView();
    void UpdateClassView(std::shared_ptr<CElement> pElement);
    void UpdateFileView();
    void UpdateFileView(std::shared_ptr<CElement> pElement);

// Managing Object Format
public:
    void FillColor(CModeler1View * pView);
    void NoFillColor(CModeler1View * pView);
    void LineColor(CModeler1View * pView);
    void LineWidth(CModeler1View * pView, UINT nID);
    void PageColor(CModeler1View * pView);

// Managing Zoom Operations
public:
    void Zoom(CModeler1View * pView);
    void ZoomIn(CModeler1View * pView);
    void ZoomOut(CModeler1View * pView);

// Load Module Operations
public:
    void LoadModule(CModeler1View * pView);

// Managing Object Selection
public:
    bool HasSelection();
    bool IsSelected(std::shared_ptr<CElement> pElement);
    bool Select(std::shared_ptr<CElement> pElement);
    bool Deselect(std::shared_ptr<CElement> pElement);
    void SelectNone();
    void DrawSelectionRect(CModeler1View *pView);

// Overridables
public:
    virtual void PrepareDC(CModeler1View * pView, CDC* pDC, CPrintInfo* pInfo);
    virtual void Draw(CModeler1View * pView, CDC * pDC);
    virtual void DrawEx(CModeler1View * pView, CDC * pDC);
    virtual void Update(CModeler1View * pView, LPARAM lHint, CObject* pHint);

// UI Handlers
public:
    virtual void OnLButtonDown(CModeler1View* pView, UINT nFlags, const CPoint& cpoint);
    virtual void OnLButtonDblClk(CModeler1View* pView, UINT nFlags, const CPoint& cpoint);
    virtual void OnLButtonUp(CModeler1View* pView, UINT nFlags, const CPoint& cpoint);
    virtual void OnMouseMove(CModeler1View* pView, UINT nFlags, const CPoint& cpoint);
};
```

Denis Duplan
Sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Coder un sine scroll sur Amiga 500

Partie 4

Cet article est le quatrième article d'une série de cinq consacrés à la programmation d'un one pixel sine scroll sur Amiga, un effet très utilisé par les coders de démos et autres cracktros durant un temps. Un exemple, dans cette intro, aussi magnifique que vintage, du groupe Miracle (1).

Dans le premier article, nous avons vu comment installer un environnement de développement sur un Amiga émulé avec WinUAE et coder la Copper list de base pour afficher quelque chose à l'écran. Dans le deuxième article, nous avons vu comment préparer une police de caractères 16x16 pour en afficher facilement les colonnes de pixels des caractères, précalculer les valeurs du sinus requises pour déformer le texte en modifiant l'ordonnée des colonnes, et mettre en place un triple buffering pour alterner proprement les images à l'écran. Dans le troisième article, nous avons vu comment dessiner et animer le sine scroll, d'abord au CPU, puis au Blitter.

Dans ce quatrième article, nous allons enjoliver le sine scroll avec quelques effets peu coûteux en cycles, car assurés par le Copper, puis rendre la main aussi proprement que possible à l'OS.

Vous pouvez télécharger l'archive contenant le code et les données du programme sur : http://www.programmez.com

Cette archive contient plusieurs sources :

- sinescroll.s est la version de base dont il sera question jusqu'à ce que nous optimisions ;
- sinescroll_final.s est la version optimisée de la version de base ;
- sinescroll_star.s est la version enjolivée de la version optimisée.

Ajouter ombre et miroir grâce au Copper

Qu'est-ce qu'une ombre portée au sud-est, sinon un bitplane qu'on affiche par-dessous lui-même, en le décalant légèrement sur la droite et vers le bas ? Et qu'est-ce qu'un miroir, sinon un bitplane qu'on continue d'afficher à partir d'une certaine ligne, mais en remontant plutôt qu'en descendant ligne à ligne dans ce dernier ?

Exposés ainsi, les effets d'ombre de miroir semble triviaux. Facile à dire ? Facile à faire ! Grâce au Copper, qui permet très simplement de modifier au début de n'importe quelle ligne l'adresse à laquelle le hardware doit lire les données du bitplane et le retard avec lequel il doit les afficher.

Comme d'habitude, commençons par définir les paramètres des effets à produire :

- SHADOW_DX et SHADOW_DY correspondent à l'ampleur de l'ombre vers la droite et vers le bas, respectivement, et SHADOW_COLOR, à la couleur de cette dernière ;
- MIRROR_Y correspond à l'ordonnée à laquelle le miroir débute, et MIRROR_COLOR et MIRROR_SCROLL_COLOR, à la couleur du fond et à la couleur du scroll dans le miroir, respectivement.

Note

Cet article se lit mieux en écoutant l'excellent module composé par Nuke / Anarchy pour la partie magazine de Stolen Data #7, mais c'est affaire de goût personnel…

1 *Sine scroll dans une intro de du groupe Miracle (https://www.youtube.com/watch?v=lOj_Y9RniTc).*

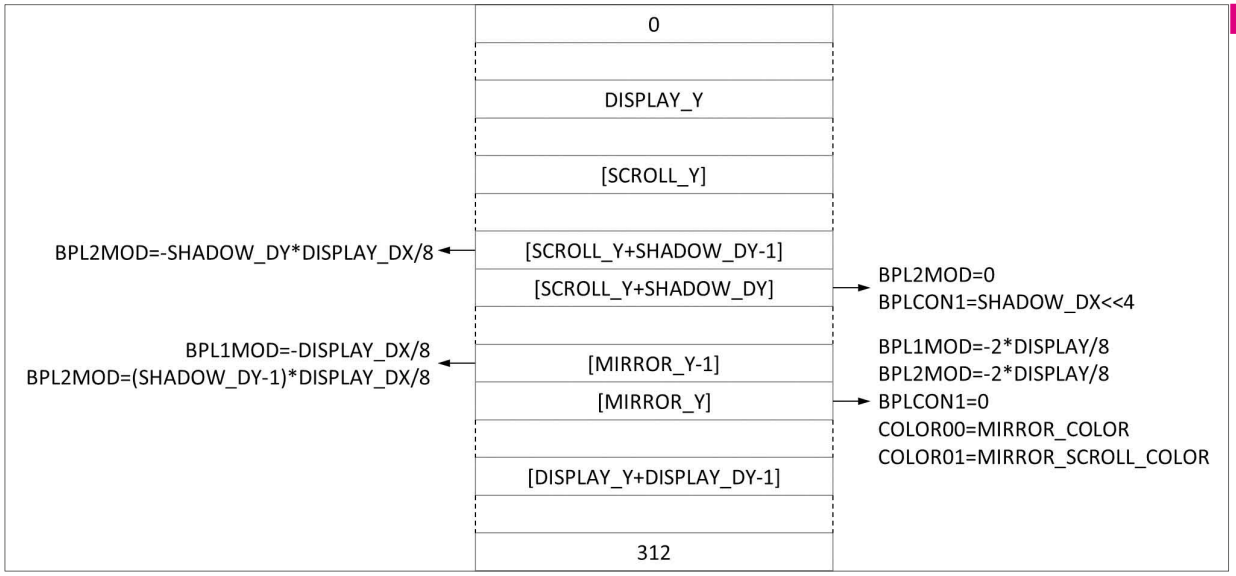
```
SHADOW_DX=2    ;Compris entre 0 et 15
SHADOW_DY=2
SHADOW_COLOR=$0777
MIRROR_Y=SCROLL_Y+SCROLL_DY
MIRROR_COLOR=$000A
MIRROR_SCROLL_COLOR=$000F
```

Essayons alors d'y voir plus clair dans la manière dont la Copper list doit se présenter. L'expérience montre que plutôt que de se lancer tête baissée dans son écriture, il vaut mieux schématiser le déroulement des opérations ligne à ligne – le Copper permet d'effectuer des MOVE en cours de ligne, mais ce ne sera pas utile ici. Pour ne pas surcharger le schéma (2), la mention d'une constante entre crochets signifie qu'il faut lui ajouter DISPLAY_Y.

L'ombre tout d'abord. Le hardware lit les données de la ligne du bitplane à afficher à l'adresse 32 bits figurant dans les registres 16 bits BLT1PTH et BPL1PTL, qu'il incrémente tandis qu'il progresse le long de la ligne. A la fin de la ligne, avant de débuter la suivante, il ajoute BPL1MOD à ces registres pour obtenir l'adresse à laquelle il commence à lire les données de la ligne suivante.

Jusqu'à présent, l'affichage se résumait à un bitplane, le bitplane 1. Nous rajoutons un bitplane 2, en indiquant au hardware que les données de ce second bitplane sont les mêmes que celle du premier :

```
move.l bitplaneA,d0
move.w #BPL1PTL,(a0)+
move.w d0,(a0)+
move.w #BPL2PTL,(a0)+
move.w d0,(a0)+
```



2 *WAIT et MOVE pour l'ombre et le miroir.*

```
swap d0
move.w #BPL1PTH,(a0)+
move.w d0,(a0)+
move.w #BPL2PTH,(a0)+
move.w d0,(a0)+
```

L'ajout d'un bitplane entraîne quelques modifications supplémentaires à la Copper list :

- spécifier le modulo des bitplanes pairs, et non plus seulement celui des bitplanes impairs :

```
move.w #BPL2MOD,(a0)+
move.w #0,(a0)+
```

- spécifier deux couleurs supplémentaires, car la palette est étendue à 4 couleurs :

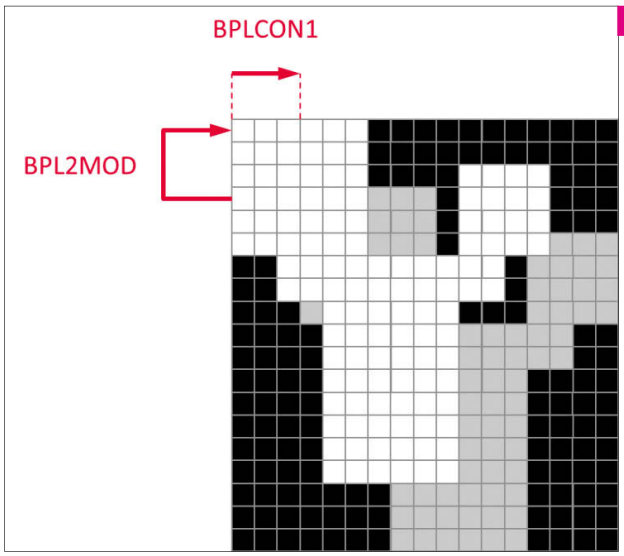
```
move.w #COLOR02,(a0)+
move.w #SCROLL_COLOR,(a0)+
move.w #COLOR03,(a0)+
move.w #SCROLL_COLOR,(a0)+
```

En passant DISPLAY_DEPTH à 2, nous modifions incidemment la valeur que le Copper stocke dans BPLCON0 pour indiquer le nombre de bitplanes, donc rien à modifier ici.

Jusqu'à la ligne [SCROLL_Y+SHADOW_DY-1], les deux bitplanes sont superposés. Le sine scroll est donc affiché avec la couleur 3, ce qui explique pourquoi SCROLL_COLOR est stocké dans COLOR03.

A partir de [SCROLL_Y+SHADOW_DY], les deux bitplanes sont décalés (Figure 3) :

- Horizontalement, en passant à SHADOW_DX dans BPLCON1 la valeur du retard avec lequel le hardware affiche les bitplanes pairs. Cette modification doit être demandée au Copper au début de la ligne [SCROLL_Y+SHADOW_DY]. Noter que SHADOW_DX ne peut dépasser 15 ; au-delà, il faut jouer sur BPL2PTH, BPL2PTL et BPLCON1 simultanément.
- Verticalement, en passant à [–SHADOW_DY*(DISPLAY_DX>>3)] le modulo des bitplanes pairs dans BPL2MOD. Cette modification doit être demandée au Copper au début de la ligne [SCROLL_Y+SHADOW_DY -1] pour affecter la ligne suivante, comme expliqué plus tôt.



3 *Décalage des bitplanes pour générer l'ombre.*

A la ligne [SCROLL_Y+SHADOW_DY], l'adresse de la ligne du bitplane 2 devient celle du bitplane 1 moins SHADOW_DY lignes. Par la suite, il est nécessaire que l'affichage du bitplane 2 se poursuive normalement, et c'est pourquoi BPL2MOD doit repasser à 0 au début de la ligne [SCROLL_Y+SHADOW_DY]. A défaut, la ligne répétée le serait indéfiniment. 3

Le miroir ensuite. Ainsi, BPLxMOD peut être mis à profit pour demander au hardware de revenir en arrière pour répéter l'affichage de bitplanes à partir d'une certaine ligne. Si DISPLAY_DX correspond à la largeur des bitplanes, alors :

- au début de la ligne [MIRROR_Y-1], passer le modulo à -(DISPLAY_DX>>3) permettra de répéter la ligne [MIRROR_Y-1] qui va être tracée à la ligne suivante [MIRROR_Y] ;
- au début de la ligne [MIRROR_Y], passer le modulo à -2*(DISPLAY_DX>>3) permettra de répéter la ligne [MIRROR_Y-2]

déjà tracée à la ligne suivante [MIRROR_Y+1], produisant un effet de miroir ;
- tant que ce modulo sera maintenu, il permettra de répéter à la ligne y la ligne tracée en 2*MIRROR_Y-1-y, ce qui perpétuera l'effet de miroir.

Après avoir choisi de faire coïncider le début de l'effet de miroir avec la fin de l'effet d'ombre, il nous reste à modifier les couleurs 0 et 3 via COLOR00 et COLOR03 au début de la ligne [MIRROR_Y] pour que le sine scroll semble se refléter dans un autre milieu. 4

Tous les MOVE que le Copper doit accomplir pour réaliser ce qui vient d'être décrit ne doivent se produire qu'au tout début de certaines lignes. Ces instructions sont donc précédées de WAIT qui instruisent le Copper d'attendre que le faisceau d'électron ait atteint ou dépassé certaines lignes.

Pour rappel, une instruction WAIT portant sur une position (x, y) à l'écran prend la forme de deux mots, un mot (y<<8)!((x>>2)<<1)!$0001 suivi d'un autre servant de masque pour indiquer au Copper sur quels bits des coordonnées la comparaison entre la position indiquée et celle du faisceau d'électrons doit porter.

En l'espèce, nous souhaitons que le Copper se contente de comparer des ordonnées. Aussi tous nos WAIT prennent-ils la forme d'un mot (y<<8)!$0001 suivi d'un mot $FF00.

Nous commençons donc par le décalage vertical de l'ombre… :

```
move.w #((DISPLAY_Y+SCROLL_Y+SHADOW_DY-1)<<8)!$0001,(a0)+
move.w #$FF00,(a0)+
move.w #BPL2MOD,(a0)+
move.w #-SHADOW_DY*(DISPLAY_DX>>3),(a0)+
```

...suivi du décalage horizontal… :

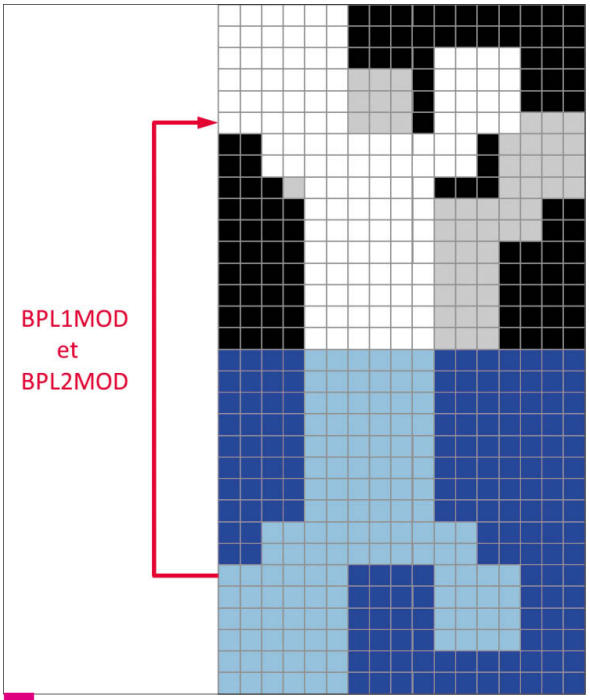
```
move.w #((DISPLAY_Y+SCROLL_Y+SHADOW_DY)<<8)!$0001,(a0)+
move.w #$FF00,(a0)+
move.w #BPL2MOD,(a0)+
move.w #0,(a0)+
move.w #BPLCON1,(a0)+
move.w #SHADOW_DX<<4,(a0)+
```

…suivi de la fin du décalage vertical de l'ombre et du démarrage du miroir… :

```
move.w #((DISPLAY_Y+MIRROR_Y-1)<<8)!$0001,(a0)+
move.w #$FF00,(a0)+
move.w #BPL1MOD,(a0)+
move.w #-(DISPLAY_DX>>3),(a0)+
move.w #BPL2MOD,(a0)+
move.w #(SHADOW_DY-1)*(DISPLAY_DX>>3),(a0)+
```

…suivi de la fin du décalage horizontal de l'ombre et de la répétition des lignes dans le miroir ainsi que de la palette qui s'applique dans ce dernier :

```
move.w #((DISPLAY_Y+MIRROR_Y)<<8)!$0001,(a0)+
move.w #$FF00,(a0)+
move.w #BPLCON1,(a0)+
move.w #$0000,(a0)+
```



4 *Répétition de lignes déjà tracées pour générer le miroir.*

```
move.w #BPL1MOD,(a0)+
move.w #-(DISPLAY_DX>>2),(a0)+
move.w #BPL2MOD,(a0)+
move.w #-(DISPLAY_DX>>2),(a0)+
move.w #COLOR00,(a0)+
move.w #MIRROR_COLOR,(a0)+
move.w #COLOR03,(a0)+
move.w #MIRROR_SCROLL_COLOR,(a0)+
```

Rendre la main à l'OS

Quand l'utilisateur clique sur le bouton de la souris, nous devons rendre la main à l'OS proprement – du moins, aussi proprement que possible, car rien ne garantit qu'il se remette de ce que nous lui avons fait subir en le coupant aussi brutalement du hardware.

Pour commencer, nous nous assurons que le Blitter n'est pas en train de travailler :

```
WAITBLIT
```

Ensuite, nous coupons les interruptions et les canaux DMA… :

```
move.w #$7FFF,INTENA(a5)
move.w #$7FFF,INTREQ(a5)
move.w #$07FF,DMACON(a5)
```

…et nous les réactivons après les avoir restaurés dans l'état dans lequel nous les avions trouvés :

```
move.w dmacon,d0
bset #15,d0
move.w d0,DMACON(a5)
move.w intreq,d0
```

```
bset #15,d0
move.w d0,INTREQ(a5)
move.w intena,d0
bset #15,d0
move.w d0,INTENA(a5)
```

Nous rétablissons alors la Copper list du Workbench. Son adresse réside à un certain offset de l'adresse de base de la bibliothèque Graphics. Pour y accéder, nous ouvrons cette librairie par un appel à la fonction OldOpenLib() d'Exec. Une fois l'adresse de la Copper list récupérée, nous demandons au Copper de l'utiliser désormais :

```
lea graphicslibrary,a1
movea.l $4,a6
jsr -408(a6)
move.l d0,a1
move.l 38(a1),COP1LCH(a5)
clr.w COPJMP1(a5)
jsr -414(a6)
```

Nous rétablissons le fonctionnement normal du système par un appel à la fonction Permit() d'Exec :

```
movea.l $4,a6
jsr -138(a6)
```

Nous libérons les espaces alloués en mémoire par autant d'appels à la fonction FreeMem() d'Exec :

```
movea.l font16,a1
move.l #256<<5,d0
movea.l $4,a6
jsr -210(a6)
movea.l bitplaneA,a1
move.l #(DISPLAY_DX*DISPLAY_DY)>>3,d0
movea.l $4,a6
jsr -210(a6)
movea.l bitplaneB,a1
move.l #(DISPLAY_DX*DISPLAY_DY)>>3,d0
movea.l $4,a6
jsr -210(a6)
movea.l bitplaneC,a1
move.l #(DISPLAY_DX*DISPLAY_DY)>>3,d0
movea.l $4,a6
jsr -210(a6)
movea.l copperlist,a1
move.l #COPSIZE,d0
movea.l $4,a6
jsr -210(a6)
```

De là, il ne nous reste plus qu'à dépiler les registres et à rendre la main :

```
movem.l (sp)+,d0-d7/a0-a6
rts
```

Reste à savoir si tout cela tient dans la trame, et à optimiser le code si tel n'est pas le cas… •

Liens utiles

WinUAE : http://www.winuae.net/

Amiga Forever : https://www.amigaforever.com/

ASM-One : http://www.theflamearrows.info/documents/ftp.html

ReqTools : http://aminet.net/package/util/libs/ReqToolsUsr.lha

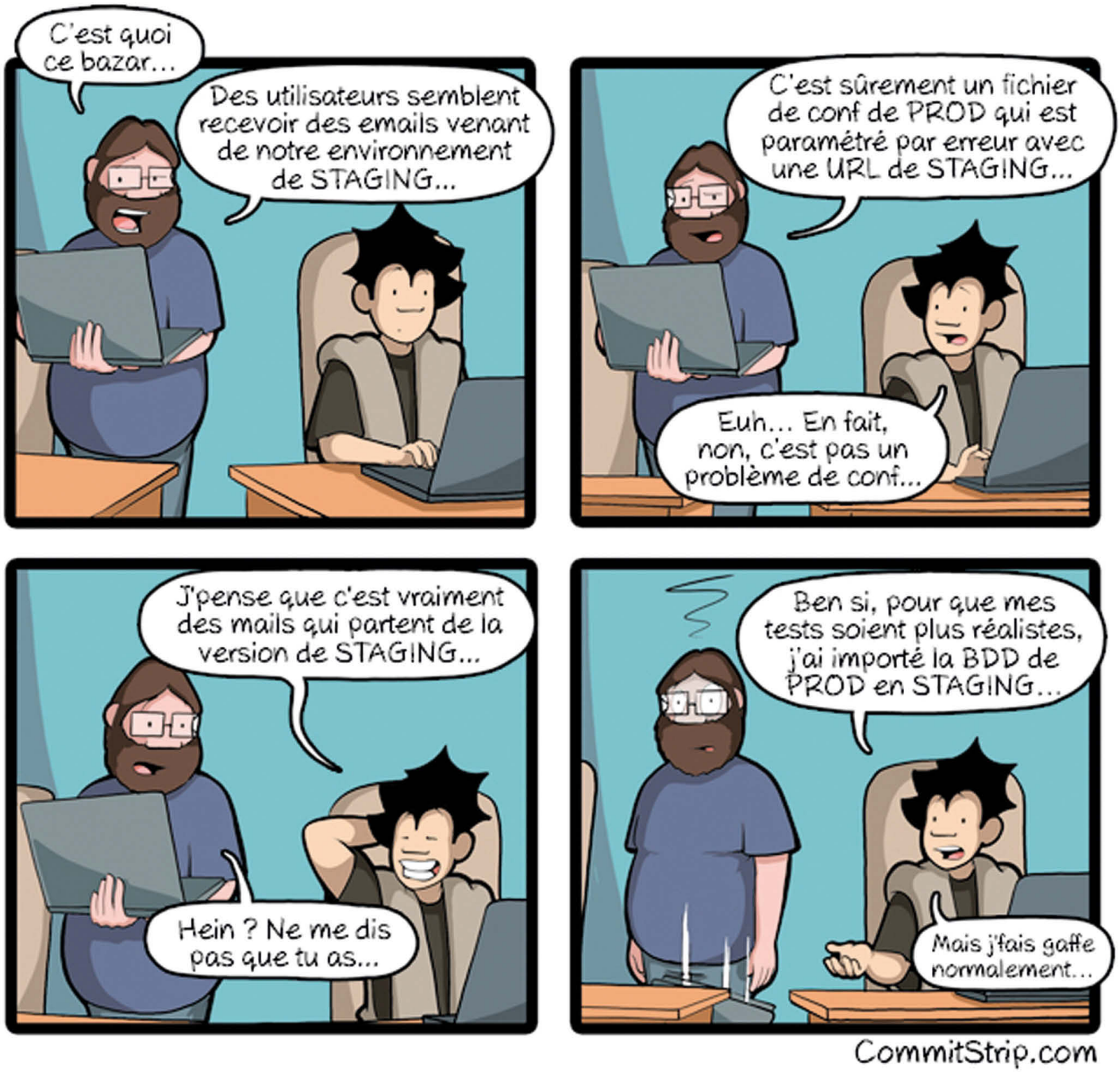
Amiga Hardware Reference Manual : http://amigadev.elowar.com/read/ADCD_2.1/Hardware_Manual_guide/node0000.html

M68000 8-/16-/32-Bit Microprocessors User's Manual : http://www.nxp.com/assets/documents/data/en/reference-manuals/M68000PRM.pdf

M68000 Family Programmer's User Manual : http://cache.freescale.com/files/32bit/doc/ref_manual/MC68000UM.pdf

Le manuel d'ASM-One : https://archive.org/details/AsmOne1.02Manual

Prod ou stagging : on ne sait plus très bien...



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, Y. Grandmontage
Nos experts techniques : B. Lane, C. Gigax, M. Bertocchi, D. Djordjedic, S. Ollivier, C. Villeneuve, T. Leriche-Dessirier, E. Shen, V. Bechu, F. Le Corre, S. Franck, C. Michel, A. Goude, B. Mesnet, C. Pichaud, D. Duplan, D. Carteau, A. Conia, R. Clark
Couverture : © Vertigo3d - **Maquette** : Pierre Sandré.
Publicité : PC Presse, Tél.: 01 74 70 16 30, Fax : 01 40 90 70 81 - pub@programmez.com.
Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN : 1627-0908 - © NEFER-IT / Programmez, mars 2018

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

NOUS RECRUTONS !

DÉVELOPPEURS
FRONT-END
BACK-END
MOBILE

Nous recrutons actuellement
des profils sur les technos java, .Net, Node.js,
Kotlin, Swift, Javascript, Angular, Php, React,
HTML 5 / CSS 3, ...

Java · Microsoft .NET · node JS · Kotlin · Swift · JS · php · HTML 5 · CSS 3

FAIRE EST LA MEILLEURE FAÇON DE PENSER

CACD2

La manufacture digitale du groupe Crédit Agricole

Notre mission est de concevoir, développer
et industrialiser des services et des produits digitaux
pour toutes les entités du groupe, autour de
problématiques fondamentales comme la finance,
l'immobilier, la santé ou encore la sécurité.

POUR PLUS D'INFORMATIONS, CONTACTEZ-NOUS :

recrutement@cacd2.fr ou cacd2.fr

WINDEV® 23 Tech Tour

VOUS ÊTES INVITÉ

SÉMINAIRE 100% TECHNIQUE

DSI
Développeurs
WebDesigners
Architectes
Ingénieurs
Start-ups...

10.000 places seulement. Réservation nécessaire

INSCRIVEZ-VOUS VITE !

MONTPELLIER	mardi 13 Mars
• GENÈVE	mardi 20 Mars
LYON	mercredi 21 Mars
STRASBOURG	jeudi 22 Mars
• BRUXELLES	mardi 27 Mars
LILLE	mercredi 28 Mars
PARIS	**jeudi 29 Mars**
NANTES	mardi 3 Avril
BORDEAUX	mercredi 4 Avril
TOULOUSE	jeudi 5 Avril
MARSEILLE	mardi 10 Avril
• MONTREAL	jeudi 12 Avril

GRATUIT ***13h45 à 17h45***

WINDEV TECH TOUR
SÉMINAIRE DÉVELOPPEMENT
WINDOWS WEB LINUX ANDROID IOS
DÉVELOPPEZ 10 FOIS PLUS VITE

PARMI LES 23 SUJETS:

BLOCKCHAIN: BITCOIN EN WINDEV • CHIFFREMENT •RGPD • IOT • RÉALITÉ VIRTUELLE • SITE WEB MOBILE FRIENDLY • SINGLE SIGN ON • UN MAGAZINE SUR MOBILE • ANALYSE DE DOCX ET DE XLSX • FRONT END BACK END LES BONNES PRATIQUES...

PC SOFT®
WWW.PCSOFT.FR