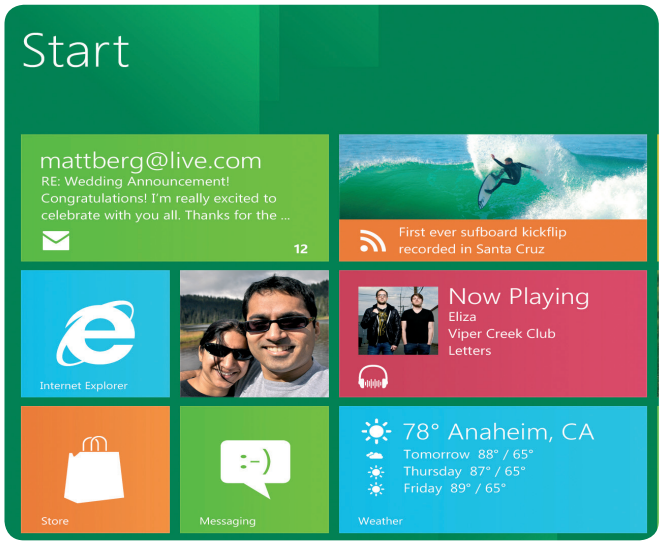
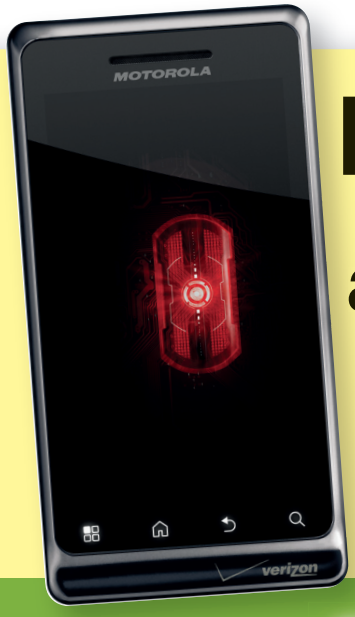
PROGRAMMEZ !
Le magazine du développement
PROgrammez !
www.programmez.com
mensuel n°145 - Octobre 2011

Les annonces de BUILD 2011

Windows 8
Azure
Visual Studio 11

Tout change !





Bien démarrer avec Android

- Ma première application
- Architecture, SDK, Java
- Faites parler vos applis

BASE DE DONNÉES

SQL Server Denali : les nouveautés

WEB

HTML 5 – GWT : le duo infernal

JAVA

Améliorer les performances d'Eclipse

Spécial Gaming

Ogre 3D

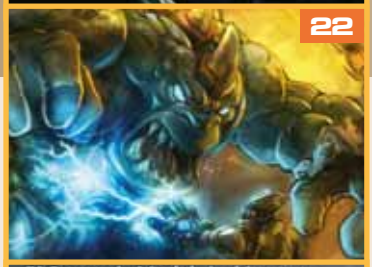
l'éclate totale !

M 04319 - 145 - F: 5,95 €

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

PROgrammez! Le magazine du développement
numéro 145 - octobre 2011

sommaire\\



Soyez parmi les millions
d'applis téléchargées
chaque jour sur Nokia Store.

Rendez-vous sur **www.nokia.fr/developpeurs**

* Là où il faut être.

NOKIA
Connecting People

éditO


©V.T.

Le Un et le Multiple

Je vois déjà les regards inquiets sur cet étrange titre. Non je ne vais pas faire un cours sur la notion monothéiste de la religion de l'Egypte ancienne, ni sur sa cosmogonie. Même s'il s'agit d'un sujet passionnant… Nous parlerons uniquement informatique. Une des questions récurrentes est de savoir s'il faut un système par type de terminal ou alors un seul système pour tous les types de terminaux.

Le rachat de webOS par HP avait pour but de proposer un unique système sur un ensemble très large de matériels : tablette, téléphone, imprimante, scanneur. Objectif : une base technique identique partout. De quoi faire des économies d'échelle sur le développement embarqué. Las, cette stratégie s'avérera un échec suite à l'annonce brutale de l'arrêt des tablettes et smartphone HP et de la vente ou de la filialisation de la division PC. Surtout, plusieurs centaines de personnes de la partie webOS risquent d'être virées.

Cette logique d'intégration, à la fois verticale et transversale, se manifeste chez plusieurs constructeurs – éditeurs. Apple a unifié son système mobile iOS en ayant un même niveau de version sur l'ensemble des terminaux mobiles : iPod, iPhone, iPad et AppleTV. Rappelons que les fondations de iOS sont celles de MacOS X et que le langage et les outils de développement sont identiques. Bien sûr, aujourd'hui, une application iOS ne fonctionne pas directement sur un OS X et inversement. Mais le rapprochement OS X – iOS a commencé, et à terme on peut penser que le modèle applicatif va devenir de plus en plus homogène et unique ; seuls les contextes d'utilisation et l'interface changeront.

Microsoft, avec son futur Windows 8 (sortie attendue d'ici 12-14 mois) n'annonce pas autre chose : Win8 est commun aux tablettes et PC, et sans aucun doute sur smartphone, même si ce point reste à éclaircir. Surtout, Win8 embarque deux interfaces : Windows classique et interface Metro provenant de Windows Phone 7.x. Selon le contexte d'utilisation, l'une ou l'autre de ces interfaces sera pertinente. Et l'éditeur a pris soin d'élargir le modèle de développement, tout en l'unifiant.
Google, avec Android, se situe grosso modo dans le même esprit que Apple et Microsoft: unification avec la v4 sur mobile et tablette. Il ne manque plus que son arrivée sur le netbook en lieu et place de ChromeOS, incompatible avec Android. Cette dualité système est critiquée depuis l'annonce de ChromeOS.

Théoriquement cette approche multi-plate-forme unifiée devrait plaire aussi bien à l'utilisateur qu'au développeur. Pour l'utilisateur, conserver une expérience identique, les mêmes repères, est un avantage important.
Du point de vue du développeur, le résultat apparaît plus ambigu. Il bénéficie d'une base technique identique, même si sur chaque type de contexte, les API, les contraintes changent. Le modèle de développement sera peu ou prou identique mais là aussi, il devra composer avec les caractéristiques de chaque terminal. On ne développe pas une interface desktop comme sur une tablette ou une box. Les fonctions, les contraintes matérielles ne sont pas les mêmes. Bref, l'impérieuse nécessité de garder une branche de code (même si les changements seront ciblés) par terminal cible demeure. Cette multiplication des branches complique les évolutions, les maintenances, les tests, et multiplie de facto les risques de bugs et d'erreurs. Et surtout, notre geek-développeur devra acheter tous les matériels cibles. Même s'il adore ça, le banquier ne sera pas forcément content.

■ François **Tonic**
Rédacteur en chef

Le magazine du développement
PROgrammez!
www.programmez.com

Editeur : Go-02 sarl, 21 rue de Fécamp 75012 Paris - diff@programmez.com.
Rédaction : redaction@programmez.com
Directeur de la Rédaction : Jean Kaminsky.
Rédacteur en Chef : François Tonic - ftonic@programmez.com. **Ont collaboré à ce numéro :** F. Mazué, T. Lellouche, S. Saurel **Experts :** P. Ognibene, P. de Saint Steban, D. Caro, A. Vannieuwenhuyze, H. Tremblay, S. Cordonnier, C. Hetier, F. Queudret, A. Verla, G. Renard..
Illustrations couverture : D.R Motorola, Microsoft
Publicité : Régie publicitaire, K-Now sarl. Pour la publicité uniquement : Tél. : 01 41 77 16 03 - diff@programmez.com.
Dépôt légal : à parution - Commission paritaire : 0712K78366 ISSN : 1627-0908. Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles Belgique. **Directeur de la publication :** J-C Vaudecrane
Ce numéro comporte un encart jeté OVH

Abonnement : Programmez, 17, Chemin des Boulangers, 78926 Yvelines Cedex 9 - Tél. : 01 55 56 70 55 - *abonnements.programmez@groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € - CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter. **PDF :** 30 € (Monde Entier) souscription exclusivement sur www.programmez.com

■ **HP** investit dans la sécurité. Pour ce faire, le constructeur a créé une division sécurité à part entière. Elle comptera un millier de personnes et regroupera les compétences HP ainsi que l'ensemble des rachats réalisés dans ce domaine (ArcSight, Fortify, etc.). La stratégie à terme est de fournir des outils unifiés de sécurité notamment sur la protection des parcs, la mise en place de politique sécurité, etc.

■ **Apple** devrait sortir son nouvel iPhone, iOS 5.0 et iCloud courant octobre. L'iPhone 5 sera doté de l'iOS 5.0 finalisé en septembre. Cette version apportera de nombreuses améliorations et nouveautés sur l'interface, les fonctions (comme des fonctions vocales largement améliorées). Elle fonctionnera aussi sur iPad.

■ Lors de la conférence Dreamforce, **Salesforce.com** a officiellement enterré le projet VMforce soutenu avec VMware. L'objectif était d'implémenter le support Java / JEE dans le PaaS maison : force.com. Mais après un an de travail, le projet n'était toujours pas arrivé à son terme. Salesforce.com change sa stratégie et mise sur le PaaS récemment racheté : Heroku.

■ **Mandriva** dévoile Mandriva 2011. C'est la première version majeure depuis le départ d'une partie des développeurs. Cette édition comporte des nouveautés appréciables comme TimeFrame pour retrouver rapidement des documents par date. On y trouve aussi RocketBar, un nouvel élément de KDE. La version introduit aussi le nouveau thème graphique : Rosa.

fOSSa 2011 : la grande conférence de l'open source à Lyon

D.R.

En dehors des salons et conférences open source, Linux ayant lieu à Paris ou encore du FOSDEM en Belgique, depuis quelques années, une conférence de haut niveau se déroule à Lyon : la fOSSa (free open source software academia). L'objectif est de recueillir, à travers des échanges libres et ouverts avec les grands acteurs de l'open source, des réponses aux questions :

- où va le domaine de l'openness ?
- qu'est-ce que les " tech people " nous préparent ?
- quelles questions et quels challenges s'annoncent dans le développement du logiciel libre, de la collaboration, du partage et de la production de la connaissance, et ce dans les domaines académique, éducatif et industriel ?

La 3e édition se déroulera du 26 au 28 octobre prochain et est co-organisée par l'INRIA, l'INSA de Lyon, IRILL et Engineering Group. HP Open Source Division et Live Projects en sont les deux sponsors.

La journée du mercredi 26 octobre 2011 abordera les relations entre l'open source et le secteur académique et les questions clés des communautés (genèse, architecture de participation, promotion…), les interventions du jeudi 27 basées sur le développement de code et la recherche, présenteront des retours d'expériences sur des méthodes innovantes de collaboration et de (re)contribution. Nous reviendrons aussi sur l'un des fondamentaux de l'open source: " l'openness ". Le vendredi se focalisera sur les nouvelles tendances qui se dessinent dans le monde de l'open source : le calcul à haute performance, le thème " mobile & personal cloud ", les questions liées au " peopleware ", la réalité mixte 3D, les futures génération de forges logicielles, le phénomène de l'imprimante 3D open source " RepRap " et le déferlement de l'open source sur le continent Africain.

fOSSa est organisée par une équipe de passionnés, qui met beaucoup d'énergie pour que ce rendez-vous soit un moment d'échange fort entre le public et les intervenants. La taille modeste de ce salon est une opportunité pour tous de rencontrer et de côtoyer durant trois jours des acteurs majeurs de l'OSS, des personnes qui définissent l'Open Source 3.0 :-) c'est aussi l'occasion de découvrir les tendances, d'échanger des bonnes pratiques, de comprendre mieux la gestion de projets de type communautaire… ALORS PROFITONS EN !

La plupart des présentations se dérouleront en anglais. Les journées fOSSa sont ouvertes à tous. L'enregistrement gratuit s'effectue sur le site: http://fossa.inria.fr

Attention, les places sont limitées.

D.R.

Advanced database technology for breakthrough applications

Laissez vos applications s'envoler...

Avec InterSystems Caché, faites décoller vos applications. Elles bénéficieront immédiatement de performances hors du commun, deviendront massivement scalable et ne nécessiteront plus d'administration fastidieuse.

InterSystems **Caché**® base de données post-relationnelle, mais aussi serveur d'application, framework Ajax, ... se fonde sur une technologie Objet avancée qui permet de construire beaucoup plus facilement des applications XML, Web Services, AJAX, Java et .NET.

InterSystems Caché est aussi une base SQL jusqu'à 5 fois plus rapide que les bases de données relationnelles classiques en accès SQL et bien plus en accès Objet !

Grâce à son Architecture de Données Unifiée unique, Caché élimine le besoin de mapping objet-relationnel, réduit les temps de développement, et facilite l'évolution et la maintenance de votre application.

Déployé sur plus de 100.000 systèmes de par le monde pour des applications de 2 à 50.000 utilisateurs, Caché est disponible sur toutes les plates-formes majeures du marché.

Nouveau: L'intégration de Caché avec JAVA – JNI permet littéralement d'exploser les performances et d'offrir enfin aux programmeurs JAVA un moteur sans compromis et digne de leurs réalisations.

Depuis plus de 30 ans InterSystems vous apporte des technologies avancées qui vous permettent de construire des applications qui font la différence.

INTERSYSTEMS

Téléchargez votre version gratuite complète InterSystems Caché - sans limite de temps: InterSystems.fr/avancee

Les Rapports DevExpress

PRESENTATION CONTROLS | REPORTING CONTROLS
BUSINESS APP FRAMEWORKS | IDE PRODUCTIVITY TOOLS

XtraReports Suite

Cross-Platform .NET Support

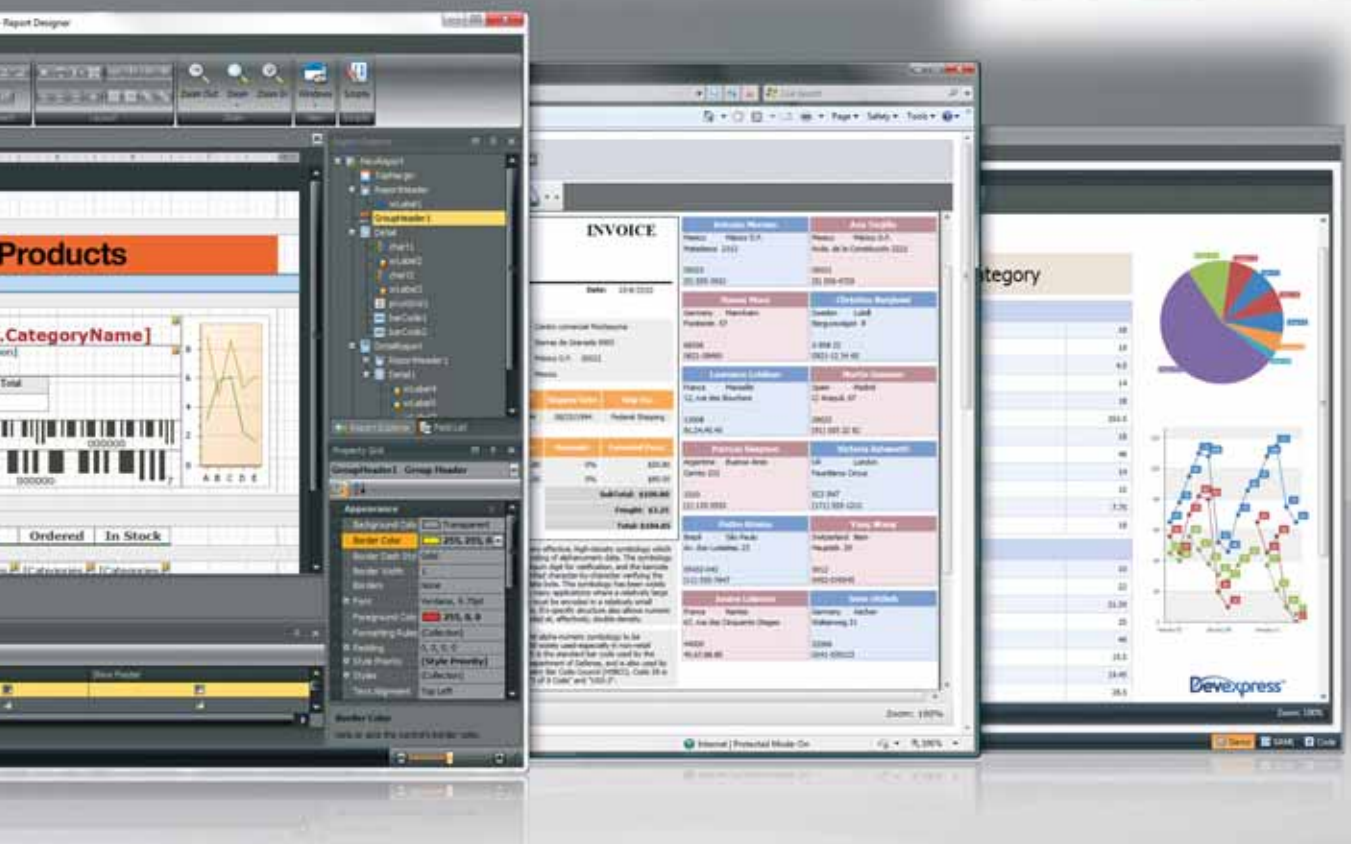
DevExpress crée des contrôles de présentation aux fonctionnalités complètes, des systèmes de rapports, des outils de productivité pour IDE et des frameworks d'applications Business pour Visual Studio .Net. Nos technologies vous aident à construire ce qu'il ya de meilleur, à avoir une vision plus claire des logiciels complexes, à améliorer votre productivité et à créer, dans le temps le plus court, des applications étonnantes pour Windows et pour le Web.

XtraReports Suite, de DevExpress, est une plateforme de Reporting de nouvelle génération pour Visual Studio. Elle vous permet de créer très rapidement des rapports professionnels, au pixel près, ciblant toutes les plateformes majeures Windows, incluant **WinForms**, **ASP.Net**, **Silverlight**, **WPF**. Au-delà de son ensemble de fonctionnalités de niveau professionnel, XtraReports Suite est livré avec un Designer de Rapports ergonomique pour l'utilisateur final, vous permettant de répondre avec la plus grande flexibilité aux exigences de Reporting.

Téléchargez aujourd'hui votre version gratuite d'évaluation et faites l'expérience de la Différence DevExpress.

www.DevExpress.com/Reporting

WIN ASP SL WPF



Devexpress™

WWW.DEVEXPRESS.COM

■ **Google** propose depuis peu une API pour son réseau social, Google+. Disponible en pré-version, elle est encore très limitée. Par exemple au 19 septembre, seul l'accès en lecture des données publiques était possible. Google+ API exige un token OAuth 2.0 ou une API key. Site : http://developers.google.com/+/api/

■ **PostgreSQL 9.1** est disponible. De nombreuses améliorations ont été introduites : mise à jour du PL-Python, support des permissions SELinux, support des « tables étrangères ». La partie réplication est modifiée comme le fait de mettre en pause une réplication esclave. Page française : http://www.postgresql.fr/

■ **Google** dévoilera lors de la conférence Goto Aarhus, un nouveau langage : DART. Il s'agirait d'un langage interprété comme Java, C# et bien d'autres. Les créateurs de DART sont des experts en programmation Web. L'objectif est clair pour DART : offrir un nouveau langage pour structurer la programmation web ! Toutes les réponses en octobre prochain.

■ **Mozilla** dévoile le projet Paladin. Il s'agit de créer une technologie dédiée aux jeux 3D sur le web. Le cœur du projet est le moteur Gladius. Ce framework travaille actuellement avec CubicVR pour le rendu. Des API seront disponibles pour la souris, la manette de jeux, etc. Le tout optimisé pour Firefox. L'équipe veut que Firefox soit le meilleur outil de jeux sur le web, rien que ça.
A voir.
Site : https://wiki.mozilla.org/Paladin

Intel propose et démontre

Intel tenait mi-septembre sa conférence développeur IDF. Le fondeur a dévoilé sa vision du futur de l'informatique, du many core partout ou presque, un accord autour d'Android… Résumé des principales annonces.

Le solaire comme énergie

Tout d'abord, Intel pousse fortement les ultrabooks et voudrait que les constructeurs y investissent largement, à l'image du Macbook Air, référence du marché. Intel a longtemps eu du mal avec ce segment à cause de processeurs mal calibrés : peu puissants ou trop consommateurs. Le fondeur met en avant la nouvelle génération de processeurs, des techniques pour un démarrage plus rapide, la connectique Thunderbold (qui arrivera enfin sur PC courant 2012). Intel compte sur Ivy Bridge pour faire exploser le marché mais ce processeur n'est pas attendu avant le premier semestre 2012, si tout va bien. Enfin, le fondeur compte sur le Haswell (2013), 3e génération des puces Core.

JavaScript boosté

Le many core est revenu sur la scène. Tout d'abord, Intel a rappelé que le marché est désormais multicore. Mais surtout, la société travaille beaucoup sur les optimisations. Ainsi, Intel a présenté un projet prometteur : Parallel JS / River Trail. Il s'agit de pouvoir exécuter du code Javascript sur plusieurs cœurs, ce qui peut, potentiellement, améliorer les performances du JavaScript, des applications, des jeux l'utilisant, etc. River Trail est une extension à Javascript pour introduire une parallélisation des données. Cela nécessite une réécriture du code, même si celle-ci serait limitée. River Trail peut servir à HTML 5, WebGL pour améliorer le rendu, l'exécution. Maintenant, il faut voir comment cette extension va s'intégrer dans les moteurs JavaScript.
Intel travaille beaucoup sur la consommation et les processeurs basse tension, sans (trop) impacter la performance. Ce projet se nomme Near Threshold Voltage Processor. Il s'agit de réduire considérablement la consommation et d'abaisser le voltage nécessaire. Cela permettrait à de nombreux terminaux, objets de fonctionner sans pour autant peser sur le réseau électrique. Une autre idée est de pouvoir directement alimenter le processeur par l'énergie solaire.
Une autre annonce, surprise, fut un accord autour du système Android et des puces x86 d'Intel. Le but est d'optimiser Android sur les architectures x86. Cette annonce concerne les processus mobiles Aton Medfield. Le but d'Intel est d'être plus présent sur les smartphones et les tablettes, marchés sur lequel ARM est très fort.
Et le système mobile Meego, soutenu par Intel, est quasi mort, sauf miracle, Nokia l'ayant lâché, Intel ne peut assumer seul son développement. Reste à avoir plus de détails sur cet accord et comprendre les concrétisations réelles. Intel n'a pas pour autant oublié Windows 8. Windows 8 sera disponible sur tablettes courant 2012 et le processeur Intel y est bien placé, même si Microsoft mise beaucoup sur ARM.
Site officiel River Trail : https://github.com/rivertrail

agenda \\

OCTOBRE

• Le 13 octobre 2011, Issy les Moulineaux, Centre de Conférences Microsoft France, **C++ Day-Microsoft & Intel**
Une journée pour découvrir les dernières innovations matérielles et logicielles autour du C++
http://msdn.microsoft.com/fr-fr/cppday

• Le 20 octobre, Paris St Lazare, 3e soirée **Meet'OSS** organisé par Smile, « Développement PHP, le defi de l'industrialisation » !
www.meetoss.com

• **Microsoft Days 2011** jusqu'au 15 novembre dans 7 villes de France - Le 4 octobre à Paris CNIT. www.microsoft.com/france/microsoft-days

NOVEMBRE

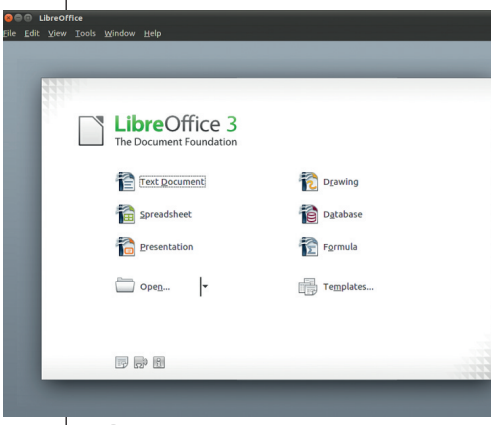
• Le 10 novembre 2011, Maison des associations, Paris13e première édition de **Drupagora 2011**. Le premier événement Européen sur Drupal dédié aux chefs de projets et aux directeurs des systèmes d'information.

ETRANGER

• Du1er au 05 octobre, USA, Los Angeles Convention Center, **Adobe Max 2011**. Inscriptions ouvertes sur : http://max.adobe.com/attend/pricing/

• Du 11 au 14 octobre 2011, USA, McEnery Convention Center de San Jose en Californie
GPU Technology Conference 2011
http://www.gputechconf.com

■ LibreOffice avance et même très bien ! Nous pouvons suivre les tribulations d'un développeur du projet via son blog : **Michael Meeks**. Celui-ci fait un constat assez édifiant des différences entre OpenOffice et LibreOffice mais aussi du travail déjà fait sur la concurrence d'OO : 678 fichiers retirés, 526 000 lignes de code supprimées, plus de 290 000 rajoutées. Mais surtout, le développeur explique une différence de plus en plus forte entre le code de Libre et OpenOffice : plusieurs millions de lignes diffèrent !
Bien entendu les nouvelles fonctions, les rajouts, modifications font fluctuer le code mais clairement, les deux projets sont désormais très différents et l'échange de code deviendra de plus en plus difficile.



Source : http://people.gnome.org/~michael/blog/2011-09-06.html

■ **VMware** dévoile Workstation 8. Cette nouvelle version de l'outil de virtualisation propose plusieurs évolutions importantes : connexion à distance (vSphere, vCenter), partage de machines virtuelles, déplacement des VM vers vSphere, une interface utilisateur revue, le support du son surround, USB 3 et une meilleure 3D. Prix : 199 dollars.

■ **Embarcadero** a dévoilé début septembre son outil FireMonkey. Il s'agit d'un outil pour créer des applications métiers pour desktop et mobile supportant à la fois la HD et la 3D, grâce à une optimisation CPU et GPU. Pour Embarcadero, tandis que Windows demeure aujourd'hui la plateforme métier standard, Mac OSX et iOS se développent rapidement. Cette croissance, combinée aux politiques naissantes dites « BYOD » (« bring your own device » : utilisez votre propre appareil), crée une forte demande et des opportunités pour les développeurs afin de concevoir des applications métier multiplateformes. Besoin auquel FireMonkey veut répondre. L'outil est inclus avec Delphi XE2, C++ Builder XO2, RAD Studio X2.

■ **Oracle** propose depuis peu Solaris 10 8/11. Cette version conçue par Sun propose des nouveautés intéressantes : migration facilitée vers ZFS, support des processeurs sparc et x86 récents, meilleures performances pour Oracle 11g. Les administrateurs bénéficient d'outils plus fins, notamment dans le reporting.

■ Accord **Oracle – Google** autour d'Android ? Pour le moment, non, le juge a simplement invité les deux parties à négocier.
Mais il y a de fortes chances que le procès ait bien lieu à partir de fin octobre prochain. Google cherche à la fois à faire retirer des mails gênants pour sa défense et estime le montant exigé par Oracle trop élevé. Pour Oracle, un accord amiable n'est peut être pas l'issue recherchée.
Si procès il y a, les conséquences, bonnes ou mauvaises pourront être importantes pour Android.

Tous les jours :
l'actu et le téléchargement
www.programmez.com

■ Immersion totale avec le casque virtuel de **Sony**, le HMZ-T1. Le casque virtuel permet de voir en 3D HD du contenu vidéo. Il est l'équivalent d'un écran de 19 mètres. Le casque possède deux écrans OLED de dernière génération. La partie audio n'est pas en reste avec la technologie Virtual Phones (son 5.1). Lancement : automne 2011, environ 800 €...

■ **PNY** a dévoilé son nouveau monstre de GPU : la XLR8 Liquid Cooled Geforce GTX 580. Sous ce nom barbare se cache un refroidissement liquide de la carte. La solution Liquid Cooled Graphics de PNY offre des performances notoires : une amélioration des températures pouvant aller jusqu'à 30%, une émission sonore réduite de 30% et une augmentation des performances de la carte pouvant atteindre10% (par comparaison avec le standard de référence graphique NVIDIA GeForce GTX 580).
Le potentiel d'overclocking de la carte s'en trouve aussi amélioré permettant ainsi aux joueurs de profiter de performances exceptionnelles. Prix : environ 560 €.

■ **G-Technology** by Hitachi propose depuis peu des disques durs 4 To pour l'A/V et les professionnels de la création ou des utilisateurs très exigeants. Cette annonce se couple avec celle d'un G-Raid 8 To en connectique Thunderbolt.
Tous les disques durs G-RAID sont préconfigurés en mode RAID 0 pour assurer un débit maximum et spécifiquement conçus pour les tâches de montage vidéo avec une excellente circulation d'air et un refroidissement optimisé.

Envie d'un drone ?

Un des gadgets les plus tendances est l'AR Drone de Parrot. Il s'agit d'un quadricoptère, donc doté de 4 moteurs. C'est un véritable mini drone multidirectionnel et équipé de deux caméras avec transmission directe des images, idéal pour des jeux, le pilotage, la réalité augmentée. Il se commande via un terminal iOS, un outil est disponible depuis début septembre (Ar. FreeFlight) mais il faut disposer d'une connexion wifi. Le plus dur est de maîtriser l'engin et de ne pas être frustré par son autonomie. Le cœur du drone est un système Linux embarqué sur processeur ARM. Des API (AR drone open API platform), disponibles pour le développeur permettent d'accéder aux fonctions de base du drone. Le tout est packagé dans un SDK. A l'heure où nous écrivons cet article, la version 1.8 est disponible : dédié à iOS (la version Android ne devrait pas tarder), mise à jour du firmware, bug fix, exemples de code et documentation. La SSII Winwise a conçu un outil de pilotage depuis une table Surface, une tablette Windows 7. Nous proposerons une série d'articles sur AR Drone très prochainement !
Site API :
https://projects.ardrone.org/

D.R

Où en est HP avec sa division PC ?

Depuis plusieurs semaines, HP hésite sur le sort réservé à sa division PC et à webOS. Après avoir annoncé l'arrêt des tablettes TouchPad et autres smartphones, HP a annoncé début septembre la fabrication d'une nouvelle série de tablette webOS. Les stocks ayant été bradés à moins de 99 euros pour la version entrée de gamme. Le sort de webOS a été au cœur des polémiques. Et son avenir reste à écrire : sera-t-il confié à une filiale indépendante ? Gardé en interne ? vendu ? Il semblerait que le système mobile resterait à HP mais les choix finaux ne devraient pas être dévoilés avant fin 2011, de quoi inquiéter clients et employés !

Sur un air d'AirPlay

AirPlay est une technologie sans fil pour diffuser du contenu audio / vidéo, mais pas seulement. Son initiateur est Apple qui l'utilise dans les dernières versions du système iOS et maintenant dans OS X Lion.
Ainsi avec une Apple TV et un iPad, il est simple de diffuser le contenu de sa tablette vers une télévision, via le boîtier Apple TV.
Aujourd'hui, grâce au succès des terminaux iOS, AirPlay s'impose peu à peu comme LA technologie sans fil pour le matériel de salon HiFi, la télévision. Le dernier salon IFA (début septembre en Allemagne) spécialisé dans les appareils HiFi, autoradio, etc. a confirmé cette tendance : de plus en plus de matériels sont compatibles AirPlay. Et nous avons l'embarras du choix des marques (Philips, Denon, Harman, Pioneer).
Seul défaut : le prix souvent élevé ! Il se murmure que des constructeurs automobiles seraient intéressés par cette technologie ! Des implémentations non officielles existent, telles que Airfoil, axStream.
AirPlay utilise le protocole UDP par le streaming audio, RTSP pour la partie réseau. Les données sont cryptées par AES.

Retour sur BUILD 2011 : la conférence développeur puissance 8

© Microsoft

Microsoft a organisé du 13 au 16 septembre dernier sa nouvelle conférence développeur remplaçant la PDC : BUILD. Cette grand'messe a attiré plusieurs milliers de développeurs du monde entier (l'entrée était facturée 3000 dollars) en Californie. Tout le monde attendait Windows 8 et nous ne fûmes pas déçus, même si toutes les questions n'ont pas été levées. Les développeurs ont pu observer le nouveau modèle de développement, la boutique à la Mac App Store d'Apple, les premières tablettes Win8, la stratégie web du groupe ou encore les prochains Windows Server 8 et SQL Server, sans oublier les nouveautés et annonces autour de Windows Azure, des outils de développement, de la virtualisation, différents SDK.

Pour le développeur .Net / Windows, BUILD 2011 pose la première pierre pour les prochaines années. Car Windows 8 aura un impact important sur le modèle de développement Windows (nouveaux frameworks, nouveaux outils, nouvelles pratiques) et il faudra quelques mois pour comprendre, apprendre et maîtriser les nouveaux paradigmes de programmation, d'interface et de fonctionnement du système. Le modèle « Windows Store » aura lui aussi un impact sur la manière de vendre et de déployer les applications. Nous avions vu la même chose avec le Mac App Store et OS X Lion. Nous allons revenir dans ce mini-dossier sur les principales annonces : Windows 8, Visual Studio 11, Windows Server 8, .Net 4.5, WinRT, les tablettes, Windows Azure, etc. Nous reviendrons dès le mois prochain en profondeur sur Windows 8 et l'ensemble des nouveautés s'annonçant pour 2012.

Avertissement !

N'installez jamais une pré-version d'un logiciel ou d'un système sur une machine de travail ou de production. Isolez toujours ces logiciels sur un disque dur séparé, un PC de test ou une machine virtuelle. Vous êtes seul responsable des pertes de données, de crash machine. Windows 8 est disponible uniquement en "clean install".

François **Tonic**

« Nous avons réinventé Windows »

Voici comment Microsoft a présenté la première sortie officielle de Windows 8 durant la conférence BUILD. Et c'est le chef de la division Windows / Windows Live qui l'a martelé : Steven Sinofsky. Totalement repensé en partant du chipset et de la volonté de s'imposer dans le monde des interfaces tactiles, le pari de Microsoft semble réussi au travers des présentations de cette première mouture de Windows 8. Un drôle de pari pour l'éditeur qui a complètement repensé ses interfaces en s'inspirant très fortement de l'expérience Windows Phone. Mais la Build 2011 ne s'est pas arrêtée à Windows 8. On y a parlé cloud, serveur, outil de développement, langages, App Store, et de bien d'autres choses ! Durant les deux sessions plénières, Microsoft a dévoilé une vision globale et intégrée, le tout centré autour de Windows, comme le montre la [Fig.1]. Un monde interconnecté, simplifiant l'usage de Mme – M. tout le monde, centré sur les usages des consommateurs et leurs nouvelles habitudes de consommation d'une information de plus en plus dense, rapide et collaborative !

Fig.1

Windows 8 pour les nouveaux matériels et usages

Immanquablement, Microsoft donne l'impression de se recentrer : renouer avec le marché, reprendre la main sur la mobilité, mettre en avant l'ubiquité. Ce n'est pas un hasard si le responsable Windows assura la présentation de Windows 8. Et même si durant BUILD, la version ARM de Win8 s'est faite discrète, elle existe et sera disponible sur cette architecture processeur (le couple Wintel est donc révolu, mais pas totalement). Microsoft aborde une nouvelle approche de son système d'exploitation en permettant une expérience utilisateur, en mode tactile ou en mode souris. La double interface (standard et Metro) est une des curiosités majeures de Win8.

Ayant eu la chance de disposer d'une Tablette Samsung (i5, 4Go DDR3, 64GO SSD) en prêt pour la semaine, mon premier rapport à Windows 8 est une vraie surprise. Fluide, réactif et intuitif, ce nouvel OS semble très prometteur et il y a fort à penser que la firme américaine à, une fois de plus, fait un pari des plus judicieux. Windows 8 reprend et s'appuie sur les points forts de Windows 7 et intègre tout son historique et ce, principalement pour la communauté développeur : vitesse, fiabilité, sécurité, compatibilité, … Windows 8 amène plus de fonctionnalités au niveau de la sécurité, un démarrage plus rapide, une plus grande autonomie, une exploitation tactile, etc.

La révolution Metro et de l'interface

Sans pour autant faire disparaître le « Desktop » bien connu des utilisateurs, les applications de type « Metro » se veulent fluides et dynamiques et apportent une dimension tactile au système et un nouveau paradigme de IHM comme sur les mobiles. La différence vis-à-vis de la concurrence se faisant notamment par l'ajout de « Charms » disponibles dans l'ensemble des applications « Metro » et permettant la mise à disposition de contrats de services. Pour exemple, le « Search » permettant ainsi de chercher aussi bien sur le système de fichier, que sur internet ou encore dans une application. Deux autres « Charms » font aussi leur apparition, « Share » qui permet de partager du contenu sur différentes plateformes type Facebook, Twitter et consorts. Ainsi que le « File Picker » qui permet de sélectionner un fichier en provenance de différents fournisseurs, pour exemple : le système de fichier, facebook images, flickr, SkyDrive … A cet effet, le support de SkyDrive est important au sein de Windows 8, il permet notamment de sauvegarder son paramétrage de l'écran « Start » afin de le retrouver sur un autre device ou bien de faire du partage de Calendrier de façon très simple entre différents utilisateurs.

Difficile de faire la liste exhaustive des annonces au vu de leur nombre, entre le support des chipsets de type ARM, le « Fast Boot », le « Secure Boot », l'amélioration de « Windows Defender », la détection du mode Broadband pour éviter les téléchargements coûteux ou encore la mise en place d'un « Windows Store » pour distribuer les applications vers l'utilisateur final, il est difficile de ne pas en oublier. Intéressons-nous donc plutôt à ce que cela changera pour nous les développeurs. Tout d'abord, le nouveau modèle de développement d'applicatifs dits « Metro ».

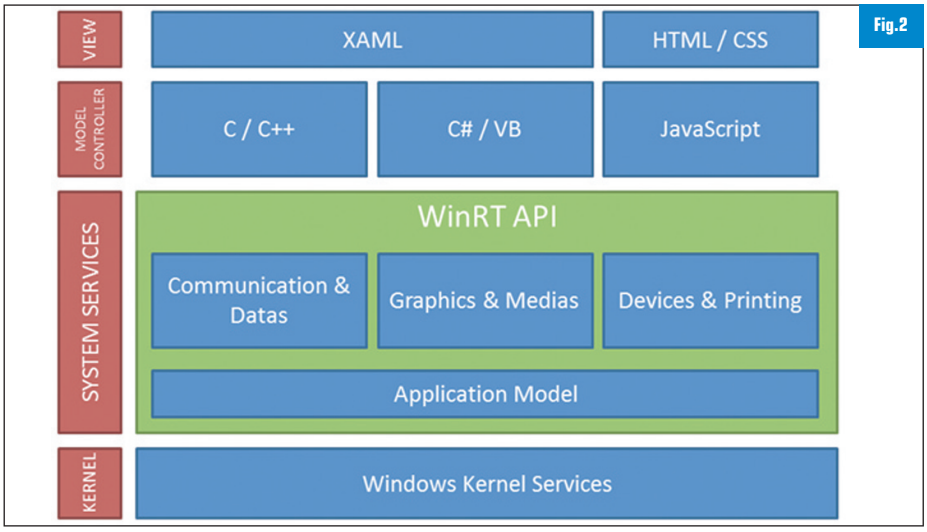


Fig.2

Blend, WindowsRT, Visual Studio 11… [Fig.2]

Comme vous pouvez le constater, le choix de la technologie de développement se fera en fonction de vos affinités. Nous pouvons cependant remarquer la présence du couple HTML/JavaScript ainsi que celle de C/C++ / XAML. L'ensemble de la WinRT API (Windows Runtime) étant exposée quel que soit le langage choisi par le développeur et permettant ainsi le dialogue avec le système d'exploitation. Pour cela, un « Visual Studio 11 Express for Windows Developer Preview » [Fig.3] est mis à disposition et permet notamment de remarquer que la version de .net sur lequel tout repose est la 4.5 ! Visual Studio 11 (édition complète disponible sur MSDN) est l'outil de développe-

*«**C'est l'histoire** d'une entreprise en pleine croissance.*
Chaque lundi, elle accueille deux nouveaux collaborateurs qui doivent attendre plusieurs semaines avant de disposer d'un poste téléphonique.
*En effet, l'opérateur traîne pour installer de nouvelles lignes et peine à augmenter la capacité du standard. **Alors qu'avec OVH...**»*

Inclus avec votre ligne SIP :

- APPELS ILLIMITÉS entre les lignes OVH partout dans le monde
- APPELS ILLIMITÉS vers les fixes en France et dans 40 pays
- Livraison en 24 heures
- Téléphones prêts à l'emploi
- Portabilité du numéro **offerte**
- Conférence téléphonique
- Renvois et redirections d'appels
- Serveur Vocal Interactif ...

Retrouvez notre brochure
dans ce numéro

Rapide

Économique

Pratique

Évolutive

La solution de téléphonie professionnelle

Ligne SIP Entreprise

Frais de mise en service ~~9,99€ HT~~ OFFERTS

+38 services inclus + **appels illimités vers FIXES**

4,99 € HT/mois

soit **5,97 €** TTC/mois

Ligne SIP Entreprise illimitée

Frais de mise en service ~~9,99€ HT~~ OFFERTS

+38 services inclus + appels illimités vers FIXES
+ appels illimités vers MOBILES

14,99 € HT/mois

soit **17,93 €** TTC/mois

Retrouvez toute la gamme, tarifs et conditions sur notre site

Plus d'infos sur : **www.ovh.com** ou **09 72 10 10 10**
Coût d'un appel local / gratuit depuis une ligne OVH

N°1 de l'hébergement Internet en Europe

ADSL | Domaines | Emails | Hébergement | Private Cloud | Serveurs dédiés | Cloud | Housing | Téléphonie | SMS & Fax

OVH.COM

ment de référence à Windows 8. Cette version propose en majorité des Templates de Projets liés à « Metro » quel que soit le langage choisi par le développeur. Blend se dote lui aussi d'une version 5 Developer Preview permettant notamment de gérer les projets de type HTML / CSS. Autre nouveauté, mais pas des moindres, la capacité de debugger en « remote machine ». Effectivement, le support « Client Hyper-V » de Windows 8 est directement exploité par Visual Studio et permet de lancer une application en mode debug directement dans une machine virtuelle. Une excellente idée.
Un petit lien à lire :
http://msdn.microsoft.com/en-us/library/windows/apps/hh465427

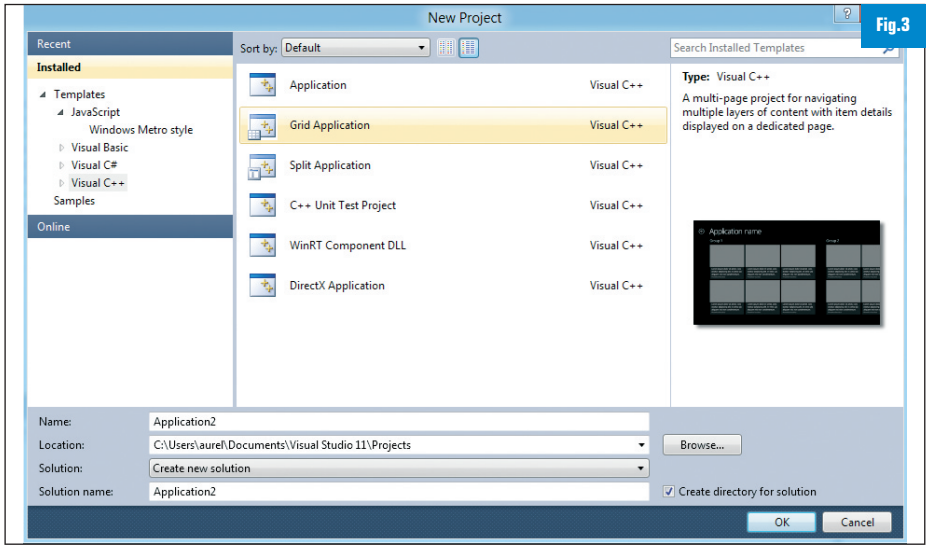


Fig.3

Plateforme et outillage Windows 8

La partie « Desktop Apps » est le modèle bien connu des développeurs à ce jour. En effet, nous y retrouvons les API Win32, le Framework .NET, Silverlight, ainsi que le couple HTML/JavaScript. Nous retrouverons donc nos applicatifs dits « lourds » développés en Win32 et WPF sur le bureau tels que nous les connaissions avec Windows 7. La nouvelle partie concerne ce que Microsoft appelle plus communément les « Metro Apps ». Ces applications ne seront pas disponibles sur le bureau traditionnel mais bien au travers du nouvel écran de démarrage et proposeront des interfaces allégées permettant facilement la consommation d'informations, a contrario du bureau et de ses applications qui permettront la création de cette information. [Fig.A]. Ces « Metro Apps » donnent naissance à une nouvelle brique native dénommée « Windows Runtime Library » (WinRT). Comprenez une « boîte-noire » qui permettra de s'assurer du bon fonctionnement des applicatifs. L'enjeu sur les tablettes tactiles est tel que Microsoft se doit de cloisonner un minimum l'utilisation de certaines API. Prenons pour exemple les accès disques, le namespace System.IO, bien connu des développeurs n'est pas disponible au travers du langage C# des applicatifs Metro. Il est cependant accessible au travers de WinRT sous une autre forme. Il est cependant techniquement possible d'utiliser System.IO dans un applicatif Metro mais il serait fort étonnant que cette utilisation soit autorisée par le processus de certification du Windows Store. [Fig.B]. En y réfléchissant, cela est logique. La première problématique du monde tablette tactile est l'expérience utilisateur et il serait fort dommageable qu'une opération bloque complètement l'interface. A cet effet, WinRT utilise beaucoup les capacités asynchrones de façon à garantir une fluidité des interfaces. La décision prise par Microsoft au niveau de WinRT est que toute opération durant plus de 50 millisecondes sera traitée en asynchrone. Cela peut paraître très contraignant pour un développeur mais c'est surtout une garantie supplémentaire d'offrir une expérience agréable. De plus, la projection de WinRT vers le développeur est multiple. Ce dernier peut effectivement choisir entre différentes technologies pour son applicatif. Tout d'abord, le couple HTML5 / CSS3 / JavaScript qui semble, a première vue, en dessous des autres possibilités. Au sens WinRT du terme, il n'en est rien. L'ensemble de l'API WinRT est projetée quel que soit le langage utilisé. De cette façon, il est possible, mais pas forcément aussi facile, d'utiliser WinRT en JavaScript. Entendez par « aussi facile » qu'il a fallu créer une syntaxe agréable au niveau du code JavaScript pour la gestion des délégués asynchrones, par exemple.
Concernant les autres possibilités, nous retrouvons notre habituel C# / VB / XAML ainsi que C / C++ / XAML. Il est aussi a noter qu'un nouveau type de projet existe uniquement pour le monde C++, les DirectX Metro Apps. Question qui fâche : quel est l'avenir de Silverlight ?

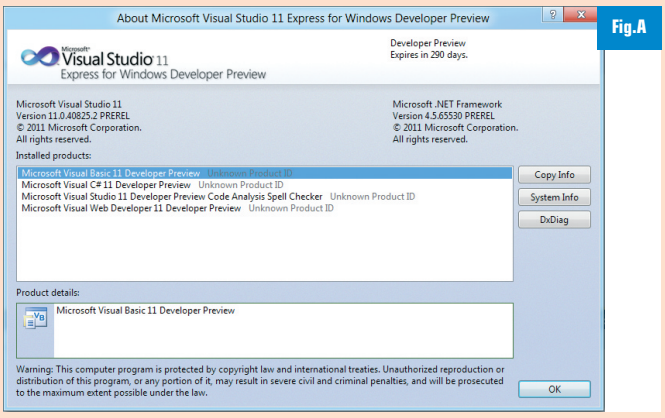


Fig.A

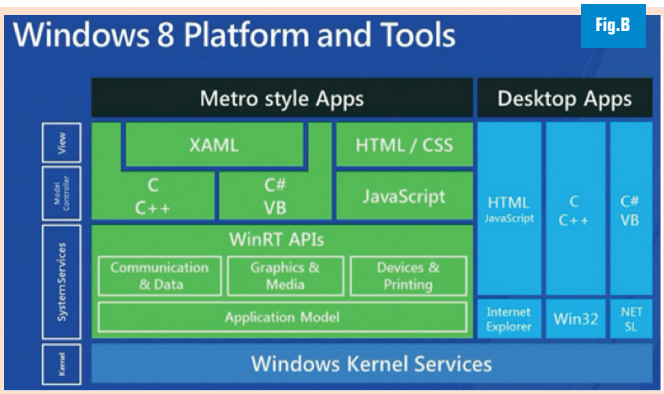


Fig.B

Windows 8 n'oublie pas le App Store

Windows 8 devient encore plus intéressant, car comme tout Shell contemporain ou de nouvelle génération, Windows 8 possèdera son propre store où les développeurs pourront déployer leurs applications afin que Microsoft se charge, tout comme sur iOS, OS X Lion, Android. Les principes de fonctionnement ne devraient pas beaucoup différer. Par contre, actuellement, seules les applications Metro seront disponibles sur le Store. Les applications pourront être payantes ou gratuites. L'installation se fera par simple clic. L'application sera soumise à Microsoft qui s'occupera de la valider et de la rendre disponible. Une fonction in-app sera disponible pour des achats, paiements à l'intérieur des applications. Reste à connaître les conditions d'accès, la tarification.

Le Cloud au coeur de Windows 8 !

Même si le cloud est discret, il n'est pas absent de Windows 8, loin de là. Nous pourrions faire un parallèle entre la stratégie iCloud d'Apple et celle de Microsoft. Comme vous l'avez remarqué, non seulement vos applications collaborent entre elles au travers de la Charms Bar mais en plus, celles-ci et son socle Windows 8 vont se charger pour vous d'aller rechercher ou publier vos données sur le cloud ! Tout ceci se passera le plus simplement du monde : Windows Live ID. Pour preuve, l'authentification qui se passe déjà au travers de celui-ci dès l'accueil ! Tous vos données, documents, photos, contacts et autres que vous souhaitez ne pas voir littéralement stockés sur votre hardware pourront se voir déportés sur le Cloud au travers du service SkyDrive jusqu'à vos settings au travers du menu "Sync PC Settings" du control panel permettant de les synchroniser (personnalisation, thèmes, langue, historique favoris ou bookmarks au sein d'IE, ...). Une pré-version du Live SDK est disponible pour les développeurs. Il propose un ensemble de contrôles et d'API permettant aux applications d'intégrer le SSO (Single Sign On) Microsoft afin d'accéder aux informations de SkyDrive, Hotmail et Messenger...

Un mot pour finir

Mais n'oublions pas qu'à ce stade, le système n'étant qu'une alpha, Windows 8 n'est qu'un Shell qui devra rapidement compter sur une communauté de développeurs et entreprises partenaires afin de créer la richesse applicative qui fait de Windows, Windows ! Le chemin est encore long ! ... Le fait d'offrir à tous les participants de BUILD une tablette Samsung va dans ce sens. Et le chemin sera encore long d'ici la sortie du système (prévue vers le 3e ou 4e trimestre 2012). Et bien des éléments peuvent encore changer. Nous n'avons pas abordé ici tous les détails sur .net 4.5, ni Windows Server 8.

Nous reviendrons dès le mois prochain sur l'ensemble des éléments décrits ici. Nous avons un an pour vous informer, expliquer et coder.

■ Aurélien **Verla** *(CTO Wygwam)* & Grégory **Renard** *(Wygwam, xBrainlab)*
et (accessoirement) François **Tonic** (Programmez)

La compilation nouvelle génération : Roslyn

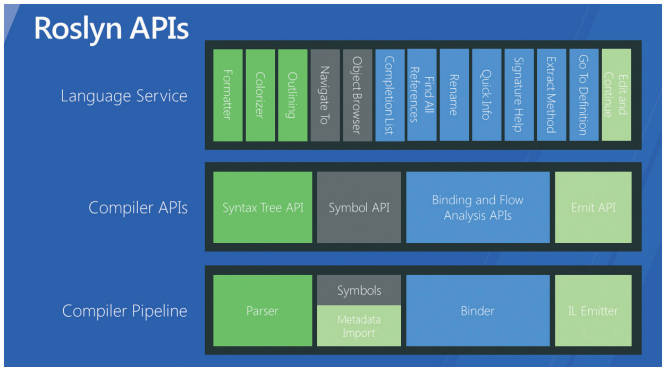
L'une des sessions les plus « geek » de cette build fût probablement celle animée par Anders Hejlsberg, architecte en chef du langage C#. Après quelques démonstrations des nouveautés apportées aux langages, les 20 dernières minutes de la session ont été consacrées à « Visual Studio Roslyn CTP ». Attention, le texte qui suit peut provoquer des hallucinations de compilations !

Le projet Roslyn, en version technique CTP, est une réécriture complète des compilateurs C# et VB sur lesquels les équipes de Microsoft travaillent depuis quelques années (l'éditeur travaille notamment sur le projet Phoenix, un nouveau modèle de compilateur optimisé et capable d'absorber les nouvelles technologies, NDLR). Le but du projet « Roslyn » est de proposer aux développeurs un « compilateur sous forme d'un service» (une sorte de CaaS), un vrai sujet pour geek.

Le cœur de Roslyn

Effectivement, la mise à disposition d'API en surcouche du compilateur permet d'imaginer nombre de scénarios allant des outils de « refactoring » à l'émission d'IL (Intermediate Language) dans votre DLL finale. Afin de mieux comprendre l'utilité de telles API, Anders a commencé ses démonstrations par le « ScriptEngine » permettant de compiler et d'exécuter dynamiquement du code C# en « string ». Exemple :

```
ScriptEngine engine = new ScriptEngine();
Session session = Session.Create();
engine.Execute("using System;", session);
engine.Execute("for (int i = 0; i < 10; i++) Console.WriteLine
(i * i);", session);
```



Une fois cette démonstration faite, il est plus facile de comprendre la nouvelle fenêtre « C# Immediate Window » permettant d'écrire et d'exécuter directement du code C#. Comprenez par là qu'il ne sera plus nécessaire de créer un projet console afin de faire quelques tests, la simple « Immediate Window » fera l'affaire. La démonstration suivante portait sur la capacité d'utiliser les API « Roslyn » pour créer des outils de refactoring autour de l'implémentation de l'interface INotifyPropertyChanged ou comment générer une dizaine de lignes de codes avec un clic de souris. Enfin, dernière démonstration sur un outil de copier/coller permettant de traduire du code C# en VB et inversement.

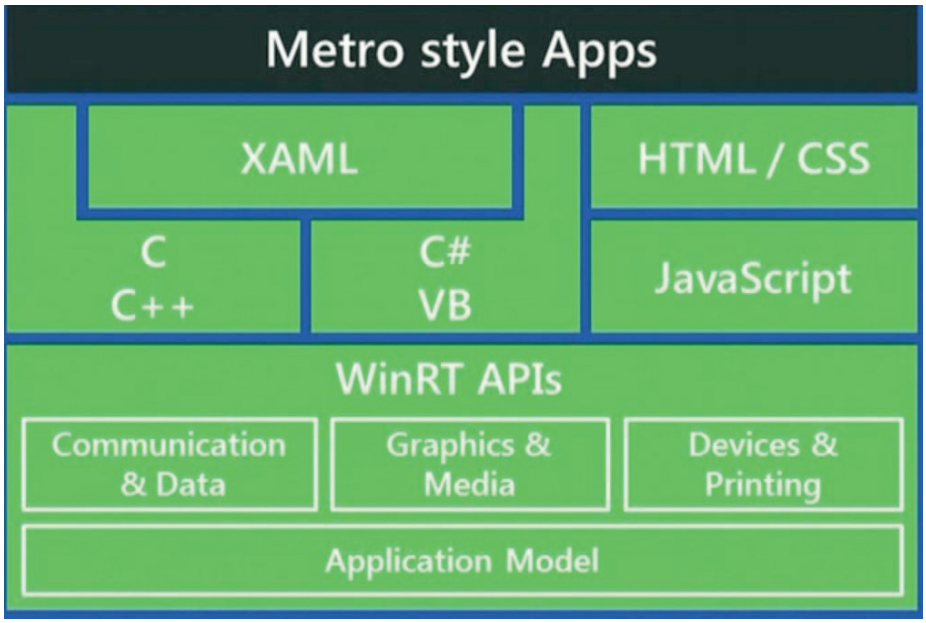
Si vous vous sentez l'âme d'un alpha testeur, je ne peux que vous conseiller de regarder cette session (http://channel9.msdn.com/events/BUILD/BUILD2011/TOOL-816T) et d'attendre patiemment la mi-octobre pour la mise à disposition de « Visual Studio Roslyn CTP ».

Enjoy !

L'architecture Win8 et WinRT

Le point de vue de Gregory Renard (alias Redo, xBrainLab, Wygwam)

Effectivement, prenez le temps de regarder le schéma plateforme et observez les deux types d'applications proposées pour Windows 8 au regard des deux modes d'affichage de Windows 8 … souvenez-vous dans mon post précédent : mode Touch "Metro" versus mode souris "Desktop" !!!



- **Métro Style Apps : Applications type Métro**

Celles-ci reposeront sur deux technologies d'interfaces utilisateurs que sont XAML et HTML/CSS pour respectivement exploiter en exécution de fond des langages comme le C, C++, C#, VB ou encore JavaScript ! La démarche est habile de la part de Microsoft et "OUI" vous avez le bon raisonnement, nous ne sommes plus tout à fait sur le .NET Framework mais bien un socle de développement appelé WinRT (Windows Runtime Library) que nous parcourrons ci-dessous. Vos acquis ne sont pas perdus, bien au contraire … XAML, C#, HTML et JavaScript prenant un rôle plus qu'important !!!

Microsoft vient tout simplement de nous présenter un nouvel environnement de développement spécifique à Windows 8 tout en nous annonçant que nous pourrons capitaliser sur nos connaissances historiques de développeur C# ou VB en vue de non plus seulement développer des applications Desktop mais aussi des applications Metro !

Pour découvrir plus en profondeur le modèle de développement, je vous invite à parcourir le site dédié au développement d'application de style Métro : http://msdn.microsoft.com/en-us/windows/apps/default et plus encore la partie Concepts et Architecture : http://msdn.microsoft.com/en-us/library/windows/apps/br211361(v=VS.85).aspx

- **Desktop Apps : Applications traditionnelles Windows**

Ici, nous retrouvons notre environnement de développement bien connu avec Win32, le .NET Framework ainsi que Silverlight. Rien à redire, hormis le fait que toute application tournant sur Windows 7 tournera de manière identique sur Windows 8 (prévoyez toutefois de bien valider votre application avant de la certifier conforme Windows 8 ;p).

Vous l'aurez ainsi compris, Microsoft nous propose un avenir en deux parties :

- Applications de Consommation (gestion d'information, de données, jeux, …) avec les applications de style Métro !
  - WP7, Windows 8 / Metro
- Applications de Création (création de contenu, données, …) avec les applications de style Desktop !

Et de ce fait enclenche progressivement une migration en douceur entre les plateformes historiques "Win32 et .NET Fx (y compris SL)" vers HTML ou XAML sur un WinRT … mais justement, qu'est-ce que ce fameux WinRT ? …

Qu'est que WinRT API ?

Pour cela, je vous invite à parcourir l'excellent post de Miguel de Icaza, en anglais, qui démystifie WinRT : http://tirania.org/blog/archive/2011/Sep-15.html

Voici en résumé des informations importantes à retenir autour de WinRT et la stratégie attachée à celui-ci :

- Microsoft profite du lancement de Windows 8 pour corriger des problèmes historiques avec Windows en apportant une nouvelle interface utilisateur digne d'une compétition dure sur le monde des tablettes et slates !
- Microsoft apporte un modèle de sécurité pour son futur Windows AppStore
- ***WinRT*** est un ensemble d'API :
  - Permettant d'implémenter le design Metro
  - A un modèle de programmation simple pour développeurs Windows
  - Exploitant XAML ou HTML / CSS en UI
  - Avec ses API qui sont toutes conçues pour être asynchrones
  - Conçues pour créer des applications Windows Store
  - Est basé sur le format ECMA 335 (idem que .NET)

WinRT intègre ainsi le nouveau système d'interface ainsi que les anciennes API Win32 ! Les autres notions importantes sont également parcourues par Miguel comme :

- la notion de "Projection" qui est le processus d'exposition des API pour ces 3 environnements : C, C++, HTML/JS et .NET (C# / VB)
- la notion d'asynchrone pour toute API, automatique pour toute exécution excédant 50 millisecondes
  - ceci en vue de garantir le bon fonctionnement des applications du Windows Store Metro et privilégier l'expérience utilisateur !
- la notion de .NET or not .NET … ? Effectivement des API ont été déplacées vers WinRT comme File I/O ou encore Sockets.

En conclusion ?

En clair, Microsoft nous apporte un troisième socle applicatif avec WinRT en plus de Win32 et .NET. Tout a été fait pour rendre son exploitation et sa compréhension le plus simple possible… sa prise en main est plus que naturelle pour tout développeur .NET … et tout ceci en vue de vous permettre de profiter du Windows Store et de la plateforme de distribution (commercialisation) de vos applications !

Les concepts du moteur 3D

Fig.1

Screenshot : une scène extraite du jeu « Torchlight », développé avec Ogre 3D, moteur Open Source

Un moteur 3D permet d'afficher des scènes en 3 dimensions. Cette définition très simple recouvre un grand nombre de concepts :

- Gestion d'une scène, des objets qu'elle contient, et des relations entre les objets
- Gestion des transformations appliquées aux objets (rotation, translation, projection)
- Gestions des matériaux appliqués aux objets (textures, shaders)
- Application d'algorithmes pour optimiser la vitesse d'affichage
- Affichage de la scène !

Quelle est la différence avec un moteur de jeu ?

Un moteur 3D ne gère pas :

- Les collisions entre objets
- Les sons et musiques
- Les entrées-sorties (gestion de la souris, du clavier, des joysticks)
- La recherche de chemins (pathfinding)
- L'intelligence artificielle
- L'affichage de boîtes de dialogue

Toutes ces fonctionnalités supplémentaires font partie du périmètre d'un moteur de jeu. Un moteur de jeu inclut souvent un moteur 3D, mais parfois il peut s'agir d'un moteur 2D (comme dans Farmville par exemple) [Fig.1].

Pourquoi utiliser un moteur 3D ?

Il y a fort longtemps (en échelle informatique ; c'est-à-dire dans les années 80-90), chaque projet utilisait son propre moteur 3D. Vu l'absence à cette époque de cartes graphiques accélératrices (1) , et même l'absence d'une API graphique normalisée comme Open GL ou Direct 3D, le moteur 3D était chargé :

- d'effectuer les calculs nécessaires (à base de projections sur les axes X, Y et Z) pour obtenir les coordonnées en espace 3D des éléments à afficher
- d'effectuer la projection des coordonnées 3D dans l'espace 2D de l'écran
- d'afficher les polygones ainsi obtenus. Ce processus s'appelle la « rasterisation » (2). Les moteurs 3D des années 80-90 effectuaient cette opération très coûteuse exclusivement avec le processeur principal (souvent Motorola m68k, puis PowerPC et x86). Les premières cartes graphiques 3D grand public, les 3dfx Voodoo, ne se chargeaient que de l'opération de rastérisation, laissant au CPU le reste des calculs.

Les moteurs 3D de cette époque étant souvent écrits en assembleur (3), ils n'offraient pas en général d'abstraction poussée des objets affichés. Ils se contentaient le plus souvent de gérer une liste de triangles pour un objet. Avec l´apparition d'Open GL, puis Direct 3D, la programmation est devenue plus aisée, puisqu'elle se fait en langage C ou C++ plutôt qu'en assembleur. Cependant, si Open GL masque la partie calculatoire d'un moteur 3D sous des matrices (de rotation, de projection), les objets manipulés restent des listes de triangles ou de polygones. Open GL n'a aucune notion de ce qu'est un « objet » au sens ou un humain peut l'entendre : une maison, une voiture, un mage niveau 35…

Gérer une scène

Une des fonctionnalités fondamentales d'un moteur 3D est de proposer une gestion de scène. Une scène représente le monde, dont seule une partie est visible à l'écran. Cette scène est constituée d'entités, qui peuvent posséder des relations entre elles, organisées sous la forme d'un arbre [Fig.2].

Les entités se décomposent souvent en plusieurs catégories :

- entités fixes, ne subissant aucun changement de structure ou de coordonnées dans le temps. Par exemple, une maison.
- Entités fixes et très nombreuses, comme des arbres
- Entités animées
- Entités pouvant entrer en collisions mutuelles
- Etc.

Cette classification est utile car elle permet au moteur 3D d'optimi-

(1) En tout cas pour le commun des mortels. SGI proposait à cette époque des stations de travail graphiques accélérées qui valaient à peu près leur poids en or.
(2) Faute d'un terme adapté en français. On pourrait utiliser pixellisation, mais le sens n'est pas exactement le même.
(3) Et en virgule fixe ! Avoir un coprocesseur arithmétique est resté un luxe à peu près jusqu'à l'arrivée des intel 486 et des motorola 68040.

ser l'affichage. Par exemple, de nombreux objets utilisant la même texture seront affichés en une seule fois (en utilisant un mode « batch »). Les cartes graphiques modernes ont en général une puissance de calcul géométrique impressionnante qui leur permet de recalculer les matrices de rotation à chaque image, mais il vaut mieux éviter de créer un million d'arbres identiques au risque de saturer la mémoire : il est préférable d'avoir une seule « instance » des coordonnées de l'objet en mémoire, et de laisser la carte graphique calculer les coordonnées des copies de cet objet.

Gérer la caméra

Les API 3D comme Open GL n'ont pas de concept de caméra. On peut la simuler avec des matrices, mais pour un être humain il est plus simple d'imaginer qu'on filme la scène. Notre caméra aura donc une position et pointera vers le point d'intérêt dans la scène. Tous les moteurs 3D proposent ce concept, souvent avec des moyens d'animer facilement la caméra (par exemple pour suivre un objet, suivre une trajectoire prédéfinie…)

Gérer les matériaux

Lorsque l'on affiche des objets en 3D, il faut qu'ils aient fière allure ! On voudra donc leur appliquer des textures (4) , des éclairages, des effets de relief, un aspect métallique ou brillant…

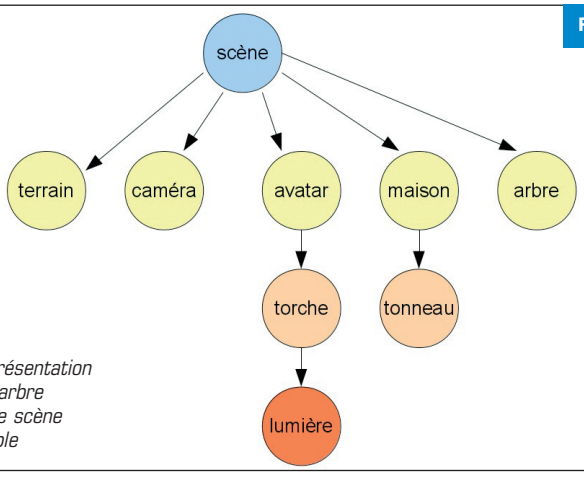
Au niveau Open GL ou Direct 3D, ces effets peuvent être gérés de deux manières :

- Via le « pipeline fixe »
- Via des « shaders », unifiés ou pas

Le pipeline fixe correspond à peu près aux versions 1.x d'Open GL, et à Direct 3D jusqu'à la version 7. Il permet d'appliquer une ou plusieurs textures éventuellement combinées avec différents effets (transparence, modulation), et également d'introduire dans la scène des sources de lumière (jusqu'à 8 en Open GL). L'ombrage est de type Gouraud, et produire des ombrages réalistes pour les objets est proche de l'impossible. A partir d'Open GL 2 et de Direct X 8, les shaders programmables font leur apparition. Il s'agit de petits programmes qui sont chargés dans la carte graphique elle-même. Ils permettent de calculer des effets d'éclairage ou de géométrie plus évolués, par exemple en tenant compte de la position de l'oeil de l'observateur par rapport à une source de lumière, et en la combinant avec une image de « relief » de la texture (5). C'est ainsi que les jeux actuels offrent de l'eau réaliste, ou des murs creusés de reliefs, sans que pour autant la scène ait été modélisée avec des milliards de triangles. La capture d'écran du jeu « Bioshock » l'illustre : le sol est constitué de quelques triangles, mais on a l'impression qu'il est extrêmement détaillé. En réalité, des shaders utilisent 3 textures (au minimum) : une pour donner la couleur d'un point de polygone, une pour donner la normale en ce point de texture, et une autre pour donner la « spécularité » (pour gérer les parties plus brillantes d'une texture) [Fig.3]. Les moteurs 3D uniformisent et simplifient l'utilisation des matériaux. Souvent, ils en proposent une gestion identique, que le programme utilise au final Open GL ou Direct 3D ou même OpenGL ES (Sur plateforme mobile). Ils proposent donc une « grammaire » unifiée pour décrire les matériaux, même si les shaders eux-mêmes doivent rester dans leur langage d'origine (CG ou GLSL ou HLSL). Ci-dessous figure un exemple de matériau dans le moteur Ogre 3D. On utilise deux textures en une seule passe. Les deux textures sont animées pour simuler un effet « d'eau » à l'ancienne, comme on pouvait l'observer dans des jeux du type Quake 3. Le paramètre « scene_blend add » permet de fusionner les deux textures par dessus le décor déjà existant (ce qui donne un effet de transparence). C'est un matériau très simple, ne dépendant ni d'Open GL ni de Direct 3D. Des matériaux plus évolués peuvent faire intervenir des shaders qui sont simplement référencés dans le matériau, mais implémentés de manière externe (en langage CG, HLSL ou GLSL, d'une syntaxe proche du langage C).

```
material Examples/WaterStream

{

 technique

 {

   pass

   {

     ambient 0.1 0.1 0.1

     scene_blend add
```

Fig.2



Représentation de l'arbre d'une scène simple

Fig.3

Image tirée du jeu Bioshock, basé sur l'Unreal Engine 3. L'effet de relief sur le sol est créé avec des shaders.

(4) Une texture est une image, généralement en 2 dimensions, qu'on peut plaquer sur un polygone. Par exemple, une photo de bois sur un objet donne l'illusion que l'objet est en bois !

(5) Ce qu'on appelle parfois du « bump mapping ». L'objet est en réalité plat, mais les shaders permettent de simuler un relief en fonction du niveau de gris d'une texture additionnelle, et de l'angle d'incidence de la lumière.

```
      depth_write off
      cull_software none
      cull_hardware none

      texture_unit
      {
        texture Water01.jpg
        scroll_anim 0.125 0
      }

      texture_unit
      {
        texture Water01.jpg
        wave_xform scroll_y sine 0 0.1 0 0.25
      }
    }
  }
}
```

En plus d'uniformiser la gestion des matériaux, les moteurs 3D avancés sont capables de réorganiser les objets et les matériaux pour optimiser l'affichage, par exemple en découpant un objet en plusieurs objets, ou en regroupant des textures entre elles pour éviter que la carte graphique ne recharge en permanence de nouvelles textures (6).

Optimiser l'affichage

Imaginez que vous avez modélisé une scène extrêmement complexe, avec des milliers d'arbres, une ville entière, des matériaux très lourds en calcul… Si vous devez tout afficher à chaque fois, votre application ne va pas être fluide. Il vous faut un moyen de réduire la quantité d'information à traiter. Les API comme Open GL ou Direct 3D effectuent de toute manière une coupe dans les objets affichés. Tout ce qui est en dehors du plan de l'écran est éliminé. Cependant, cette élimination n'intervient que dans la phase finale de l'affichage, après la projection et juste avant la rastérisation. Elle est donc coûteuse, même pour une carte graphique haut de gamme.

(6) Ce qu'on appelle un « texture atlas ». Cette astuce est très utilisée sur les plateformes mobiles, qui disposent de peu de mémoire texture et encore moins de bande passante entre le processeur principal et le processeur vidéo.
(7) Surtout lorsqu'ils sont implémentés sous forme de shader, dans les moteurs modernes comme le Cry Engine.

Il faut donc mettre quelque chose de plus pour accélérer l'affichage. Dans ce domaine, il n'existe pas une solution qui marche à tous les coups, mais plutôt de grandes familles d'algorithmes pour des situations particulières :

- Si votre scène se passe surtout en intérieur, avec un horizon « bouché » par des murs et avec des espaces assez restreints, la famille des BSP (Binary Space Partition) est adaptée. Il s'agit de séparer l'espace en un ensemble de demi-plans. L'arbre binaire résultant peut être parcouru rapidement pour tester la visibilité d'un objet (la vaste majorité des objets ne sera pas visible car masquée par d'autres éléments ; et le BSP permet de le déduire très rapidement)
- Si votre scène comprend beaucoup d'objets de taille raisonnable, avec de grands espaces vides, un algorithme du genre Octree sera plus adapté. L'algorithme Octree découpe l'espace en cubes imbriqués de taille dégressive. Pour savoir si un objet doit être affiché, il suffit de tester d'abord les cubes les plus gros. Si un « gros » cube n'est pas visible, alors tous les objets contenus dans le cube sont invisibles également.
- Si votre scène se passe dans un immense espace extérieur, vous voudrez plutôt jouer sur le niveau de détail des objets en fonction de leur distance à l'observateur (algorithmes de LOD). Certains algorithmes de LOD sont implémentés sur le processeur central (x86 par exemple), mais les plus récents utilisent des « geometry shaders », fonctionnant sur la carte graphique.

Les moteurs 3D viennent avec des implémentations pour ces différents cas, ce qui est une chance, car les algorithmes ne sont pas vraiment triviaux (7).

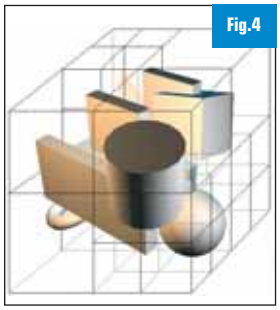


Fig.4

Représentation d'un octree (tiré du site http://www.flipcode. com/archives/Introduction_To_Octrees.shtml). On essaie de localiser les objets dans l'un des cubes, ce qui permet de déterminer rapidement ce qui est visible ou non.

Outils divers

En dehors de la batterie d'algorithmes évoqués jusqu'ici, un moteur 3D fournit toujours une librairie de calcul matriciel. En effet, même si la carte graphique peut calculer les matrices de transformation de la plupart des objets, il subsiste des opérations qu'il faut faire « manuellement » sur le processeur. Par exemple, la gestion de la caméra, ou la gestion de la sélection des objets dans une scène.

Conclusion

Même s'il est toujours possible de construire un produit en utilisant directement des API de (relativement) bas niveau comme OpenGL ou Direct 3D, il est souvent préférable de s'appuyer sur un moteur 3D afin de s'abstraire de détails d'implémentation non triviaux.
Il existe de nombreux moteurs 3D, les plus aboutis étant réalisés en C++, comme Ogre 3D ou Irrlicht. Il existe quelques produits en Java, comme jMonckey. Enfin, beaucoup de moteurs 3D sont intégrés dans des moteurs de jeu complets, comme Crystal Space pour les produits Open Source, ou l'Unreal Engine, le Cry Engine, ou Unigine pour les produits commerciaux sur PC et consoles, Unity, Shiva ou Unreal sur mobiles, pouvant coûter plusieurs centaines de milliers de dollars.

■ Pascal **Ognibene**
Directeur technique - Valtech

La 3D par l'exemple : Ogre !

Une des façons les plus intéressantes d'aborder la 3D est d'essayer de programmer un jeu vidéo. Cependant, le programmeur assez aventureux pour se lancer devra absorber un ensemble de connaissances non négligeables.
Un moteur 3D permet de faciliter les choses. Dans cet article, nous développerons un exemple simple qui pourra servir de base, en nous appuyant sur le moteur Ogre 3D.

Qu'est-ce-donc qu'Ogre 3D ? Un moteur 3D, écrit dans un C++ élégant, fonctionnant sur Windows, Linux, Mac, iPhone, et peut-être bientôt Android. Pourquoi en C++ ? Bien que Java et C# permettent d'obtenir de très bonnes performances pour certains types d'applications comme les sites WEB, dans le domaine de la 3D le C++ reste le principal langage utilisé. Il permet en effet de maîtriser l'allocation mémoire, d'éviter les saccades de l'animation dues au ramasse-miettes, et aussi de descendre aux niveaux d'optimisation les plus extrêmes, en mixant aisément du code SSE, MMX ou assembleur. Or, en 3D chaque image par seconde compte !

Donc, pour cet article, vous allez devoir écrire un peu de C++. Que le lecteur se rassure, pas beaucoup, et Ogre 3D étant très bien écrit et documenté, c'est même plutôt agréable ! Du reste, dans un vrai gros projet, le cœur sera écrit en C++, et la partie spécifique au jeu dans un langage de script comme LUA ou Python par exemple.

Ogre 3D est uniquement un moteur 3D. Il se contente donc de gérer l'affichage de scènes en 3D. Il ne gère ni les sons, ni les collisions, ni même les joysticks, souris ou clavier. Pour cela, il faut introduire des librairies supplémentaires. Dans cet article, nous en utiliserons une pour gérer le clavier et la souris.

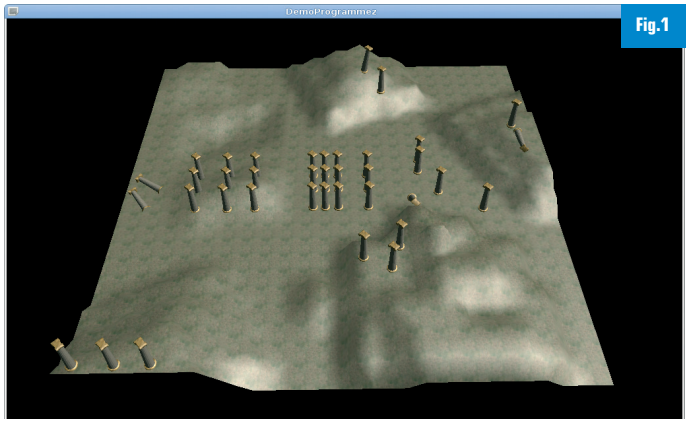


Fig.1

Copie d'écran de la scène que nous allons afficher

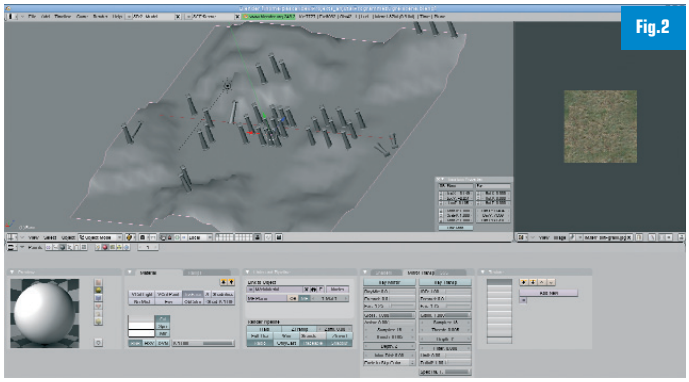


Fig.2

Une copie d'écran de blender montrant la scène à afficher dans Ogre.

Qu'allons-nous réaliser ?

Nous allons écrire ensemble un programme qui permet d'afficher une scène 3D « généraliste ». Par généraliste, nous voulons dire que nous n'allons pas utiliser d'algorithme hautement optimisé pour afficher par exemple des paysages immenses à la façon du jeu Oblivion, ou encore des scènes d'intérieur à la Doom 3. Au contraire, nous allons considérer que tous les objets affichés dans notre scène sont égaux devant le moteur 3D. Ça n'est pas une manière très optimisée de gérer une scène, mais elle suffira pour commencer [Fig.1].

Dans notre scène, nous allons afficher un terrain sous la forme d'un gros « mesh » (objet constitué de triangles), ainsi qu'une collection de colonnes vaguement grecques éparpillées sur le terrain. Nous pourrions tout à fait construire une scène beaucoup plus complexe, et le lecteur peut essayer : il suffit de modéliser les objets et de les insérer dans la scène.

Bouger la souris permet de modifier le point de vue de la caméra dans notre scène. Appuyer sur les touches « page up » ou « page down » permet de déplacer la caméra.

Mais avant de commencer…

Il nous faut parler de la chaîne de production du contenu. En clair, l'ensemble des outils que vous allez devoir mettre en œuvre pour afficher quelque chose dans votre monde virtuel.

En premier lieu, il va vous falloir un modeleur 3D, qui va vous permettre de créer les objets 3D à afficher, ainsi que leur aspect. Les plus utilisés dans le domaine des jeux vidéo sont 3DS Max, Maya, Softimage XSI… Pour notre part, nous allons utiliser Blender 2.49, un excellent outil open source connu par ailleurs dans le monde de l'image de synthèse (1) [Fig.2].

Blender permet donc de créer les objets que l'on souhaite afficher dans notre monde en 3D. On peut également affecter des textures aux objets. Par contre, à la différence d'outils intégrés comme ceux dont on peut disposer avec des moteurs de jeux comme l'Unreal Engine, on ne visualise pas en temps réel dans le modeleur les matériaux tels qu'ils seront finalement appliqués dans le jeu…(2)

Exporter en Ogre XML

Une fois un objet créé, il faut encore l'exporter pour pouvoir l'utiliser dans Ogre. La première étape consiste à l'exporter dans un format XML spécifique grâce à un script Python à installer dans votre

(1) Blender a été utilisé pour différents courts métrages comme Big Buck Bunny mais également pour certains effets spéciaux de Spiderman 2.
(2) Sauf en utilisant le moteur de jeu intégré à Blender. Il y a des travaux en cours pour l'associer à Ogre ou à Crystal Space, ce qui permettrait effectivement d'avoir un rendu « comme dans le jeu » mais dans le modeleur 3D lui même.

répertoire $HOME/.blender. Ce script peut être téléchargé depuis cette page : http://www.ogre3d.org/tikiwiki/Blender+Exporter sur laquelle vous trouverez également les instructions d'installation et d'utilisation. À noter que ce script permet également d'exporter des objets animés à l'aide d'armatures, mais nous n'utiliserons pas (encore) cette possibilité dans cet article [Fig.3].

Exporter au format Ogre binaire

Ogre ne sait pas directement utiliser le format XML. Il est donc nécessaire de transformer le fichier en un fichier binaire optimisé pour la taille et les performances. On peut le générer à l'aide de l'outil OgreXMLConverter qui vient avec le SDK :

```
>OgreXMLConverter  objet.xml
```

produira un fichier binaire .mesh que Ogre peut charger.

Exporter une scène

Dans un moteur 3D, il est possible de créer un à un les objets à afficher. C'est utile lorsque l'on veut afficher des objets temporaires, par exemple les étincelles de votre sort de boule de feu niveau 5, mais pour la géométrie « statique », il est indispensable de pouvoir construire le niveau dans un outil séparé.

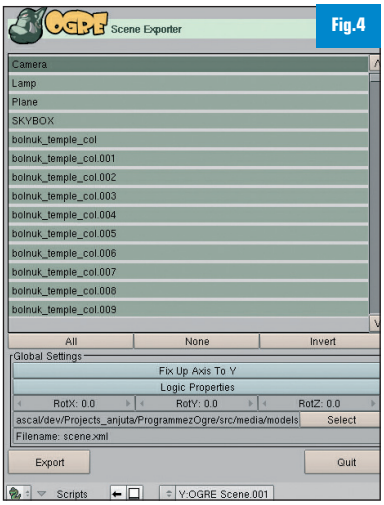


Fig.4

Nous avons choisi d'utiliser également Blender pour créer notre scène. C'est très simple : il suffit de créer un ou plusieurs objets, puis de les dupliquer avec ALT+D dans la scène. ALT+D permet de dupliquer l'objet mais en le gardant lié à son « datablock », c'est-à-dire aux sommets et textures de l'objet original, tandis que SHIFT+D dupliquerait complètement l'objet, y compris les données graphiques – donnant de fait naissance à un deuxième objet complètement indépendant du premier. Une fois les objets correctement disposés et orientés, on peut exporter la scène à l'aide d'un deuxième script disponible à l'adresse :

http://www.ogre3d.org/tikiwiki/Blender+dotScene+Exporter

Il faut sélectionner tous les objets à exporter : [Fig.4]

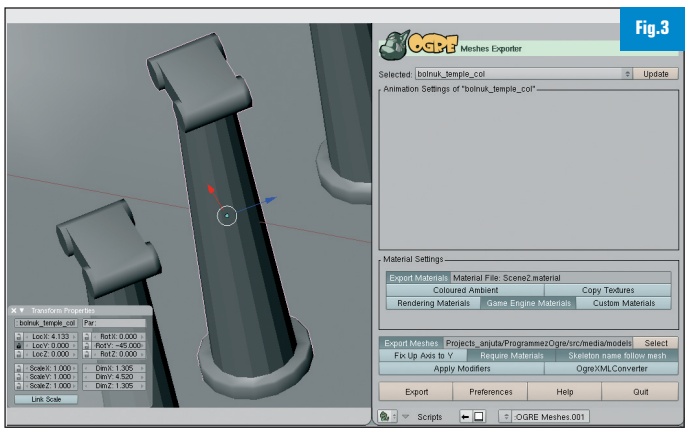


Fig.3

Copie d'écran de l'exporteur d'objets Ogre. Ici, on va exporter la colonne surlignée de rose.

En cliquant sur le bouton « export », un fichier XML est généré contenant les informations sur chaque objet (ou entité) de la scène :

```
<scene formatVersion="1.0.0">
  <nodes>
    <node name="bolnuk_temple_col.027">
      <position x="-19.463234" y="0.470225" z="0.713756"/>
      <quaternion x="0.000000" y="-0.382683" z="0.000000"
         w="0.923880"/>
      <scale x="1.000000" y="1.000000" z="1.000000"/>
      <entity name="bolnuk_temple_col.027"
         meshFile="bolnuk_temple_col.mesh"/>
    </node>
...
   </nodes>
</scene>
```

On notera que même si les entités possèdent chacune un nom et des cordonnées différentes, elles sont toutes associées au même mesh – parce que nous avons dupliqué les objets avec ALT+D.

Ce fichier XML peut être lu par Ogre. Enfin, presque… Parce que Ogre ne connaît pas ce format, mais le code pour le charger est simple à écrire et est fourni avec cet article, très fortement inspiré de celui qu'on peut trouver sur le site de Ogre.

Et si on commençait les choses sérieuses ?

Nous avons vu comment produire des objets à afficher, et comment créer une scène intégrant de multiples objets. Passons maintenant à la programmation. Tout d'abord, puisque nous sommes en C++, il va nous falloir compiler notre code. Le code fourni avec l'article vient avec une suite de scripts générés avec les autotools pour compiler sous Linux. Créer un exécutable est très simple :

```
>./configure;make;cd src;./programmezogre
```

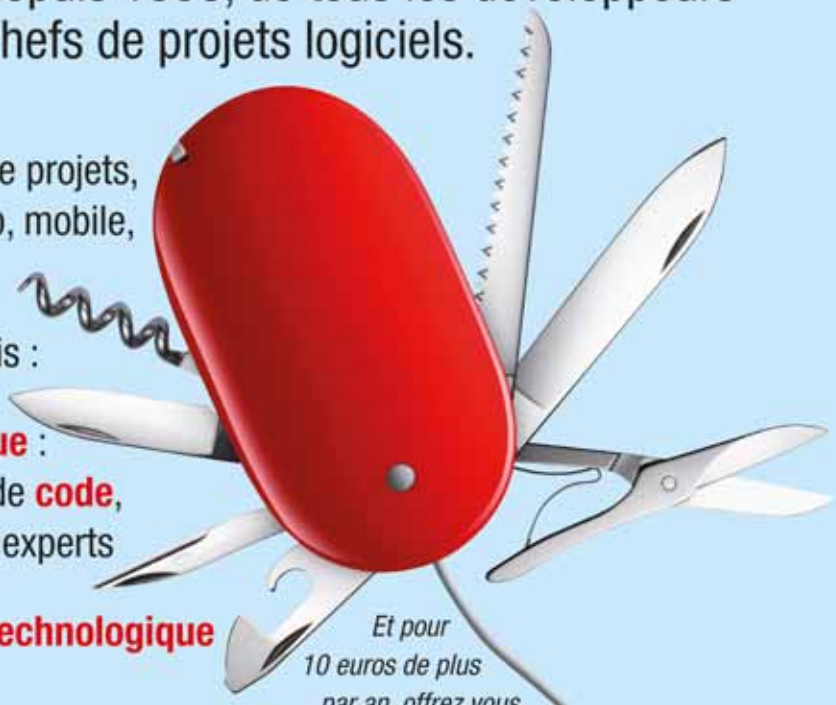
Sous Windows, vous pouvez utiliser Visual C++ ou MinGW ou Code Blocks… Une solution Visual C++ 10 devait être associée à cet article, mais des problèmes de link de dernière minute l'ont empêché : le lecteur pourra reconstituer lui même l'environnement nécessaire avec les sources mis à disposition (les seules dépendances étant les librairies Ogre et la librairie OIS, qui sont livrées avec le SDK Ogre). Comme tout programme C++, nous commençons avec une fonction main dans le fichier main.cc :

```
1. #include <OgreException.h>
2.
3. #include "GameManager.h"
4. #include "IntroState.h"
5.
6. #if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
7. #define WIN32_LEAN_AND_MEAN
8. #include "windows.h"
9. INT WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR str
CmdLine, INT )
10. {
11. #else
12. int main( int argc, char **argv )
13. {
14. #endif
```

TOUT SAVOIR POUR TOUT FAIRE

Développez votre savoir-faire !

Programmez ! est le magazine de référence,
depuis 1998, de tous les développeurs
et chefs de projets logiciels.



Code, gestion de projets,
développement web, mobile,

Programmez ! est à la fois :

- votre **outil pratique** :
  articles de **code**,
  par les meilleurs experts
- votre **veille technologique**

Et pour 10 euros de plus par an, offrez vous l'accès illimité à toutes les archives et numéros en format électronique.

Abonnez-vous à partir de 4€ seulement par mois

ABONNEZ-VOUS EN LIGNE

www.programmez.com

Toutes les offres sont en ligne

☐ OUI, je m'abonne

à retourner, avec votre règlement à :
Groupe GLI, 17 chemin des Boulangers 78926 Yvelines cedex 9

☐ *Abonnement 1 an : 49€ 11 numéros par an au lieu de 65,45€, prix au numéro (*)*

☐ *Abonnement intégral :1 an au magazine + archives : 59€ (*)*

☐ *Abonnement 2 ans au magazine : 79€ (*)*

(Tarif France métropolitaine)*

☐ M. ☐ Mme ☐ Mlle Société .....

Titre : ..... Fonction : ..... Adresse mail .....

NOM ..... Prénom .....

N° ..... rue .....

Complément .....

Code postal : ..... Ville .....

☐ *Je joins mon réglement par chèque à l'ordre de PROGRAMMEZ* ☐ *Je souhaite régler à réception de facture* (écrire en lettres capitales)

```
15.
16.     GameManager *gameManager = GameManager::getSingletonPtr();
17.
18.       try
19.       {
20.           gameManager->startGame( IntroState::getSingleton
Ptr() );
21.       }
22.       catch ( Ogre::Exception& ex )
23.       {
24. #if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
25.         MessageBox( NULL, ex.getFullDescription().c_str(),
"An exception has occurred!", MB_OK | MB_ICONERROR | MB_TASKMODAL );
26. #else
27.         std::cerr << "An exception has occured: " << ex.get
FullDescription();
28. #endif
29.       }
30.
31.       delete gameManager;
32.       return 0;
33. }
```

De la ligne 1 à la ligne 12, on inclut les headers spécifiques à Ogre, et on se débarrasse des spécificités d'une application Windows avec l'appel à WinMain. A la ligne 20 se situe le cœur du programme… Un appel à la méthode startGame d'une mystérieuse classe GameManager. Cette classe n'est pas fournie par Ogre. Elle incorpore la logique de base permettant de gérer des états – indispensables dans un jeu vidéo. Visualisez le scénario suivant :

- je lance le jeu vidéo
- un premier écran s'affiche pour le développeur du jeu
- un deuxième écran s'affiche pour l'éditeur du jeu
- un troisième écran s'affiche, avec une barre de progression, pour le chargement initial
- un quatrième écran affiche l'écran d'accueil
- depuis cet écran, je peux charger une sauvegarde, ou démarrer une nouvelle partie, ou quitter…
- etc. La succession des écrans n'est autre qu'une machine à état.

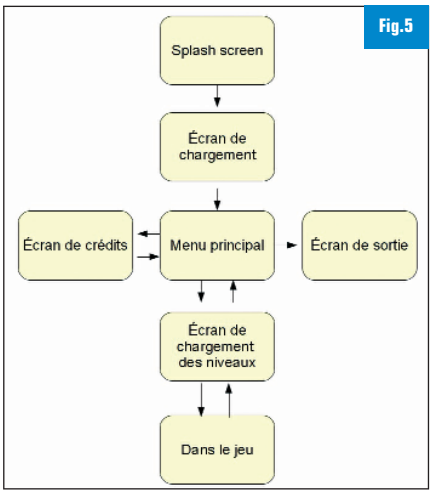


Fig.5

Cette machine à état permet de stocker :

- l'état de la caméra,
- le type de projection souhaité (perspective ou orthographique),
- la façon doit on doit traiter les événements de clavier et de souris dans cet état particulier,
- des informations spécifiques à l'état. Par exemple, inutile de conserver les informations sur le joueur dans un état qui ne sert qu'à sauver la partie [Fig.5].

Les états peuvent s'empiler ou se dépiler. Lorsqu'on dépile l'état initial, on sort du programme !

La classe GameManager (dans le fichier GameManager.cc) gère :

- l'empilement et le dépilement des états (méthodes pushState et popState)
- l'état initial du jeu
- la configuration du jeu
- les entrées-sorties (grâce à la librairie OIS dont nous parlerons plus loin)

La partie intéressante de cette classe se situe dans la méthode startGame :

```
 void
1. GameManager::startGame (GameState * gameState)
2. {
3.   mRoot = new Root ();
4.   mIntroState = IntroState::getSingletonPtr ();
5.   this->setupResources ();
6.
7.   if (!this->configureGame ())
8.     {
9.       throw
10.     Ogre::Exception (Ogre::Exception::ERR_INTERNAL_ERROR,
11.               "Error - Couldn't Configure Renderwindow",
12.               "Example - Error");
13.       return;
14.     }
15.
16.   mInputMgr = InputManager::getSingletonPtr ();
17.   mInputMgr->initialise (mRenderWindow);
18.
19.   mInputMgr->addKeyListener (this, "GameManager");
20.   mInputMgr->addMouseListener (this, "GameManager");
21.
22.   this->changeState (gameState);
23.   unsigned long lTimeLastFrame = 0;
24.
25.   while (!bShutdown)
26.     {
27.       unsigned long lTimeCurrentFrame = mRoot->getTimer()->get
Microseconds();
28.       unsigned long lTimeSinceLastFrame = lTimeCurrentFrame -
 lTimeLastFrame;
29.       lTimeLastFrame = lTimeCurrentFrame;
30.
31.       mInputMgr->capture ();
32.       mStates.back()->update (lTimeSinceLastFrame);
33.       mRoot->renderOneFrame ();
34.      Ogre::WindowEventUtilities::messagePump ();
35.     }
36. }
```

A la ligne 3, on récupère une référence sur l'objet racine de la scène, justement appelé « Root ». Il va nous donner accès à un timer précis à la micro seconde pour mesurer le temps de dessin de chaque image (3).

En ligne 5, on définit les ressources de l 'application, ce qui revient à

(3) En fait, il s'agit du temps nécessaire pour dessiner l'image mais aussi pour gérer la logique de jeu. Dans notre cas, la logique est à peu près absente donc le temps est exclusivement constitué par le temps d'affichage de la scène. Avec une carte graphique récente, vous pouvez obtenir plusieurs milliers d'images affichées par seconde sur une scène simple comme celle-ci !

charger un fichier de configuration et à ajouter l'ensemble des chemins qui y sont définis dans les chemins gérés par Ogre. Ogre sera ensuite capable de charger des objets comme des textures et des meshs dans ces emplacements. Dans le cadre de cet article, on a défini un dossier « media » à l'intérieur du dossier source, qui contient toutes les ressources nécessaires : le lecteur pourra aller jeter un œil au fichier resources.cfg et à la documentation d'Ogre pour comprendre ce qu'il contient. On définit ensuite ligne 19 des « listeners » pour recevoir les événements clavier et souris via la librairie OIS (la classe GameManager hérite d'une classe abstraite de la bibliothèque OIS pour pouvoir consommer ce type d'événements).
Entre la ligne 25 et 35 est définie une boucle sans fin : c'est la boucle principale du jeu. A chaque itération, on calcule le temps écoulé depuis le dernier passage dans la boucle, indispensable pour avoir des animations avec une vitesse constante même si la machine est très lente ou au contraire très rapide. En ligne 31 on capture les événements souris et clavier. En ligne 32 on met à jour la scène pour l'état courant (c'est dans cette opération update que l'on va animer les objets, la caméra, etc.). En ligne 33 on affiche la scène (ce qui se traduira par l'envoi transparent de commandes Open GL ou Direct X selon la plateforme d'exécution). Enfin, en ligne 34 on s'assure de bien récupérer les événements souris et clavier. Tout ceci est très simple. La vraie complexité se situe dans la classe IntroState.cc, qui est l'unique état que nous gérons dans cette application.

```
1. void IntroState::enter()
2. {
3. mRoot          = Root::getSingletonPtr();
4. mSceneMgr     = mRoot->getSceneManager( "ST_EXTERIOR_CLOSE" );
5. mCamera        = mSceneMgr->createCamera( "IntroCamera" );
6. mViewport     = mRoot->getAutoCreatedWindow()->addViewport
( mCamera );
7. Ogre::String sceneName = "Scene.xml";
8.   CDotScene scl;
9. scl.parseDotScene( sceneName,
                      "General",
                      mSceneMgr,
                      NULL,
                      "");

10. mCamera->setPosition (0, 60, 60);
11. mCamera->lookAt (0,0,0);
12. mCamera->setNearClipDistance(1.0);
13. mCamera->setFarClipDistance(10000-99);
}
```

Dans l'opération enter, nous construisons la scène. Il faut pour cela :

- choisir le gestionnaire de scène, ligne 4. Ogre intègre différents gestionnaires, généralistes comme celui que nous utilisons, ou très spécialisés, comme le « paging scene manager », dédié à des scènes d'extérieur de grande taille (à la façon du jeu Oblivion). Il est aussi possible de développer son propre *scene manager*, pour des usages très spéciaux (par exemple, un jeu en 2D accéléré par la carte graphique).
- Créer une caméra qui permettra de visualiser la scène.
- Créer un Viewport ligne 6. Il est possible d'avoir des ViewPorts multiples pour une scène, ce qui permettra par exemple de découper l'écran en plusieurs zones de dessin indépendantes.
- Ligne 9, on charge la scène XML et on crée la scène. Tout se fait en une seule ligne, mais le code sous-jacent est complexe, nous allons le détailler plus loin.
- Le reste du code, de la ligne 10 à la ligne 12, permet de placer la caméra dans l'espace et de s'assurer qu'elle regarde à un endroit qui contient un objet – ici l'origine du monde, (0,0,0).

Comment charger une scène ?

Ogre n'intègre pas en standard de moyen de charger une scène complète. Il sait par contre charger des objets de type MESH, des textures, et des matériaux (description de l'aspect d'un objet).
Il nous faut donc charger la scène à la main ! Heureusement, le format .scene qui est exporté par le plugin Blender vient également avec un exemple de code de chargement. Nous avons réutilisé ce code dans le cadre de l'article, avec quelques petites modifications.
Pour charger le fichier XML, nous avons utilisé un analyseur appelé TinyXml. Écrit en C++, il ne fait que quelques centaines de lignes et est largement suffisant pour nos besoins. Il est directement intégré sous forme source dans le projet. TinyXml est à son tour utilisé par notre chargeur de scène dans le fichier SceneLoader.cc.
Un fichier .scene peut contenir de nombreux types d'objets (4), comme des sources de lumière, une caméra, mais surtout, des entités. Les entités sont les objets que nous souhaitons afficher dans notre scène. Dans cet extrait de la méthode parseDotScene :

```
1.while( XMLNode ) {
2.    // Process the current node
3.    // Grab the name of the node
4.    Ogre::String NodeName = XMLNode->Attribute("name");
5.
6.    // First create the new scene node
7.    Ogre::SceneNode* NewNode = pAttach
8.      ->createChildSceneNode( m_sPrependNode + NodeName );
9.    Ogre::Vector3 TempVec;
10.   Ogre::String TempValue;
11.
12.   // Now position it...
13.   XMLPosition = XMLNode->FirstChildElement("position");
14.   if( XMLPosition ) {
15.     TempValue = XMLPosition->Attribute("x");
16.     TempVec.x = Ogre::StringConverter::parseReal(TempValue);
17.     TempValue = XMLPosition->Attribute("y");
18.     TempVec.y = Ogre::StringConverter::parseReal(TempValue);
19.     TempValue = XMLPosition->Attribute("z");
20.     TempVec.z = Ogre::StringConverter::parseReal(TempValue);
21.     NewNode->setPosition( TempVec );
22.   }
23.
24.   // Rotate it...
25.   XMLRotation = XMLNode->FirstChildElement("quaternion");
26.   if( XMLRotation ) {
27.     Ogre::Quaternion TempQuat;
28.     TempValue = XMLRotation->Attribute("x");
29.     TempQuat.x = Ogre::StringConverter::parseReal(TempValue);
```

(4) Et il peut également être étendu via des attributs spéciaux à renseigner dans Blender. Par exemple, pour ajouter des objets qui ne servent que pour les collisions, ou définir des portails permettant de changer de niveau, ou encore des zones actives qui déclenchent un dialogue…

```
30.     TempValue = XMLRotation->Attribute("y");
31.     TempQuat.y = Ogre::StringConverter::parseReal(TempValue);
32.     TempValue = XMLRotation->Attribute("z");
33.     TempQuat.z = Ogre::StringConverter::parseReal(TempValue);
34.     TempValue = XMLRotation->Attribute("w");
35.     TempQuat.w = Ogre::StringConverter::parseReal(TempValue);
36.     NewNode->setOrientation( TempQuat );
37.   }
38.
39.   // Scale it.
40.   XMLScale = XMLNode->FirstChildElement("scale");
41.   if( XMLScale ) {
42.     TempValue = XMLScale->Attribute("x");
43.     TempVec.x = Ogre::StringConverter::parseReal(TempValue);
44.     TempValue = XMLScale->Attribute("y");
45.     TempVec.y = Ogre::StringConverter::parseReal(TempValue);
46.     TempValue = XMLScale->Attribute("z");
47.     TempVec.z = Ogre::StringConverter::parseReal(TempValue);
48.     NewNode->setScale( TempVec );
```

Le parseur parcourt récursivement l'ensemble des éléments du fichier. Lorsqu'il trouve un élément (entité ou source de lumière ou autre). Ligne 8 l'appel à createChildSceneNode permet de créer un nœud dans la scène. Ce nœud cependant n'est pour l'instant associé à rien. Dans les lignes suivantes, on positionne le nœud en X,Y,Z, on lui applique une échelle et une rotation selon les attributs trouvés dans le fichier XML. A ce stade, le nœud n'est toujours qu'un objet abstrait, qui n'entraînerait pas d'affichage à l'écran. Il nous faut donc lui attacher une entité.

```
XMLEntity = XMLNode->FirstChildElement("entity");
   if( XMLEntity ) {
     Ogre::String EntityName, EntityMeshFilename;
     EntityName = XMLEntity->Attribute( "name" );
     EntityMeshFilename = XMLEntity->Attribute( "meshFile" );

     // Create entity
     Entity* NewEntity = mSceneMgr->createEntity(EntityName,
 EntityMeshFilename);

...
     NewNode->attachObject( NewEntity );
```

Cette entité est créée :
- en chargeant le fichier MESH associé
- en créant une entité et en lui attachant le mesh
- en attachant cette entité au nœud. Ouf !

Pourquoi a-t-on besoin d'autant d'opérations pour créer un simple objet ? En fait, l'arbre de scène que gère Ogre est abstrait. Un nœud peut subir des transformations géométriques mais être attaché à un élément non affichable, par exemple une zone de collision. Un MESH peut être utilisé par plusieurs entités, auquel cas elles partagent sa « géométrie » – c'est-à-dire la description des sommets et matériaux du mesh. Utile pour économiser la mémoire de la carte graphique !

Interagir avec la scène

Nous pouvons charger une scène et la visualiser. Mais rien ne bouge : c'est un peu ennuyeux. Il serait plus intéressant de pouvoir déplacer la caméra. Pour ce faire, nous devons détecter des événements liés au clavier et à la souris. La librairie OIS va nous aider à le faire. Nous avons créé la classe InputManager suivante :

```
class InputManager : public OIS::KeyListener, OIS::MouseListener {
…
} ;
```

qui hérite des classes abstraites permettant de consommer respectivement des événements de clavier et des événements de souris. La classe InputManager se doit d'implémenter certaines méthodes : keyPressed, keyReleased, mouseMoved, etc. Ces méthodes sont appelées par OIS lorsqu'un événement se produit. La classe InputManager n'est pas strictement indispensable. Nous pourrions implémenter directement les interfaces OIS dans nos états par exemple. Mais elle permet de distribuer les événements sur des consommateurs multiples, ce qui peut être utile. En résumé :
- OIS détecte un événement clavier, souris ou joystick
- OIS invoque une classe qui implémente les méthodes de rappel nécessaires (voir le code de InputManager.cc pour plus de détails)
- La classe InputManager va relancer l'événement auprès de tous les consommateurs enregistrés, ce qui permet de découpler le code de la manière de consommer les événements

La classe GameManager s'enregistre pour recevoir ces événements. Par contre, chaque fois qu'elle va recevoir un événement, elle va le retransmettre à l'état courant :

```
bool

GameManager::keyPressed (const OIS::KeyEvent & e)

{

  mStates.back()->keyPressed (e);

  return true;

}
```

Avec l'exemple ici pour un événement de touche clavier pressée. L'état courant (dont le pointeur est retourné par mStates.back()) doit donc lui même implémenter une action spécifique pour cet événement. Dans notre cas, nous n'avons qu'un seul état (5), IntroState. Jetons donc un coup d'oeil à l'implémentation de keyPressed dans le fichier IntroState.cc :

```
void IntroState::keyPressed( const OIS::KeyEvent &e )

{

 // if the UP or DOWN key is pressed, move the camera along its
 direction vector

 if (e.key == OIS::KC_PGUP) {
```

(5) Rappelons que dans un jeu nous aurions de nombreux états différents, gérant potentiellement les événements et l'affichage de différentes façons.

```
    deltaCameraY = -1;

  }

  else if (e.key == OIS::KC_PGDOWN) {

    deltaCameraY = 1;

  }

}
```

On voit donc qu'en pressant la touche « page up » on va faire bouger la caméra dans un sens, et en pressant la touche « page down » on va la faire bouger dans l'autre sens. Mais où cela se passe-t-il exactement ? Dans la méthode update !

```
void IntroState::update( unsigned long lTimeElapsed )

{

  if(deltaCameraY != 0) {

    Vector3 direction = mCamera->getDirection();

    direction *= (deltaCameraY * lTimeElapsed)/100;

    mCamera->move(direction);

    deltaCameraY = 0;

  }

}
```

Cette méthode est appelée juste avant de dessiner la scène via Open GL ou Direct X. On récupère le vecteur direction de la caméra – c'est-à-dire la direction vers laquelle elle pointe. Puis on lui ajoute un delta modulé par le temps écoulé, en micro secondes, depuis le dernier dessin de la scène, afin d'éviter des accélérations ou des ralentissements intempestifs de l'animation. Enfin, on appelle mCamera->move() qui va ajouter le vecteur déplacement à la position actuelle de la caméra. Avec ce code, on peut faire reculer et avancer la caméra… Mais on ne peut pas la faire pivoter pour découvrir le monde. Il nous faut gérer en plus les événements souris :

```
void IntroState::mouseMoved( const OIS::MouseEvent &e )

{

  double relativeX = ((double) e.state.X.rel) / 200;

  double relativeY = ((double) e.state.Y.rel) / 200;



  Quaternion q1(Degree(relativeX), Vector3::UNIT_Y);

  Quaternion q2(Degree(relativeY), Vector3::UNIT_X);

  mCamera->rotate(q1*q2);

}
```

On reçoit dans cette méthode un déplacement relatif de la souris en pixels, sur les coordonnées X et Y. Nous convertissons ces deux valeurs en deux angles de rotation sur les axes Y et X, en passant par des quaternions (6). Puis nous appliquons à la caméra une rotation résultant de l'application des deux quaternions (le lecteur avisé notera que la surcharge de l'opérateur * permet de rendre indolore le chaînage des quaternions ou des matrices de rotation).

Récapitulons

Nous pouvons modéliser et afficher une scène, la complexité n'en étant limitée que par votre capacité créatrice et la puissance de votre PC. La caméra est gérée d'une manière primitive mais suffisante pour comprendre la gestion des événements. Nous n'avons pas d'objets animés ni de détection des collisions ; mais tout cela pourra être ajouté par la suite.

Avant de se quitter

Pour utiliser les exemples fournis, vous devrez installer un certain nombre de dépendances.

Sous GNU/Linux :

- compilateur C++ (gcc)
- librairie OIS installée avec les headers de développement. Elle est disponible sous forme de paquets sur les distributions Debian récentes, et probablement aussi sur beaucoup d'autres versions de Linux.
- Ogre 1.7.2 ou + installé. Le mieux est de l'installer depuis les sources, ce qui nécessite l'installation de librairies supplémentaires (headers X-Windows, headers Open GL, entre autres). Référez-vous au site d'Ogre 3D pour des informations complètes.
- GNU make

Les sources sont livrées avec un script configure qu'il suffit d'exécuter, puis de suivre par « make ».

Sous Windows :

- Visual C++ 10 (la version gratuite marchera très bien)
- Ogre 3D installé (la distribution binaire à télécharger sur le site d'Ogre va très bien. Elle intègre également OIS et d'autres librairies)

Sous Mac :

- le rédacteur n'a pas de Mac… Mais les dépendances doivent être assez proches de celles pour Linux. Voir sur le site d'Ogre 3D.

Conclusion

Ogre est un moteur 3D très puissant que nous n'avons fait qu'effleurer dans cet article. Il contient aussi des fonctionnalités pour gérer des animations, des particules, des trajectoires d'objet ou de caméra, des « overlays » qui permettent d'afficher en surimpression un score ou une mini carte… La courbe d'apprentissage est raide, mais le résultat en vaut la peine.

■ Pascal **Ognibene**
Valtech - Directeur Technique

(6) Nous n'avons pas la place dans cet article d'expliquer pourquoi les quaternions sont préférables à des matrices de rotation. Disons juste en bref qu'ils permettent de minimiser l'accumulation d'erreurs de calcul.

Faire du GWT en HTML5 !

HTML5 est un mot à la mode, il a remplacé Ajax et le web 2.0 ! Il est vrai que cela apporte de nouvelles fonctionnalités dans nos applications, dont une bonne partie est déjà disponible dans la plupart des navigateurs récents (Firefox, Chrome, Opéra, Safari, et même Internet Explorer dans sa version 9).

GWT (Google Web Toolkit) permet de créer des applications web en compilant notre code java en JavaScript spécifique pour chaque navigateur. Comme GWT utilise l'HTML, le JavaScript et le CSS, c'est tout naturellement qu'il est possible d'utiliser toutes les fonctionnalités de l'HTML5 avec GWT, grâce notamment au mécanisme de JSNI (JavaScript Native Interface). Mais depuis les dernières versions de GWT, un certain nombre de fonctionnalités sont directement intégrées dans le cœur du GWT. Depuis la version 2.2, nous pouvons utiliser les balises Audio, Vidéo et Canvas disponibles. Dans la version 2.3, les fonctionnalités de LocaleStorage et de SessionStorage sont disponibles. Enfin, la dernière version 2.4 apporte le Drag And Drop et la Géolocalisation. Bien sûr, il est possible d'utiliser les propriétés CSS3 dans nos applications. Pour illustrer les possibilités de l'HTML5 en GWT, nous allons créer une petite application de planning avec des Post-it que l'on peut placer sur trois colonnes (Todo, In Progress, Done), toutes les tâches seront sauvegardées et chargées depuis le LocalStorage, et enfin nous ajouterons un camembert qui reflétera l'avancement de nos tâches (le nombre de tâches à faire en rouge, en cours en orange et réalisées en vert)

Création des trois colonnes

Commençons par créer un tableau (Board) où l'on va ajouter trois colonnes. Pour les créer, nous allons utiliser le FocusPanel car il nous permet d'avoir accès aux événements de Drag and Drop et nous insérons à l'intérieur des éléments de type FlowPanel qui vont nous permettre de placer tous nos Post-it. Pour cela, créons une classe Board en UiBinder :

```
<!DOCTYPE ui:UiBinder SYSTEM «http://dl.google.com/gwt/DTD
/xhtml.ent»>
<ui:UiBinder xmlns:ui=»urn:ui:com.google.gwt.uibinder»
 xmlns:g=»urn:import:com.google.gwt.user.client.ui»>
 <ui:style>
   html,body {
     height: 100%;
   }
   .board {
     width: 100%;
     height: 100%;
   }
   .column {
     float: left;
     width: 32%;
     min-height: 500px;
     height: 100%;
     border: thin dashed maroon;
     border-radius: 5px;
     position: relative;
     margin: 2px;
   }
   .title {
     color: maroon;
     font-size: 14px;
     text-align: center;
     border-bottom: thin dashed maroon;
     margin: 0;
     padding: 3px;
     display: block;
     width: 99%;
   }

   .header {
     display: inline-block;
     font-size: 18px;
     width: 100%;
     text-align: center;
   }
   .container {
     width: 100%;
     position: absolute;
     top: 30px;
     bottom: 0;
     left: 0;
     right: 0;
   }
 </ui:style>
 <g:HTMLPanel height=»100%» width=»100%»>
   <span class=»{style.header}»>Planning Task Board</span>
   <g:Button ui:field=»newTask»>New task</g:Button>
   <g:Button ui:field=»deleteTask»>Drag here to delete</g:Button>
   <div class=»{style.board}»>
     <div class=»{style.column}»>
       <span class=»{style.title}»>To-do</span>
       <g:FocusPanel ui:field=»todoContainer» styleName=»{style.
container}»>
         <g:FlowPanel ui:field=»todo»></g:FlowPanel>
       </g:FocusPanel>
     </div>
     <div class=»{style.column}»>
       <span class=»{style.title}»>In Progress</span>
       <g:FocusPanel ui:field=»progressContainer» styleName=»{style.
container}»>
         <g:FlowPanel ui:field=»progress»></g:FlowPanel>
       </g:FocusPanel>
     </div>
     <div class=»{style.column}»>
       <span class=»{style.title}»>Done</span>
       <g:FocusPanel ui:field=»doneContainer» styleName=»{style.
```

```
container}»>
          <g:FlowPanel ui:field=»done»></g:FlowPanel>
        </g:FocusPanel>
      </div>
    </div>
  </g:HTMLPanel>
</ui:UiBinder>
```

Et la classe Java :

```
public class Board extends Composite {

   private static BoardUiBinder uiBinder = GWT.create(BoardUi
Binder.class);

  interface BoardUiBinder extends UiBinder<Widget, Board> {
  }
  @UiField Button newTask;
  @UiField FlowPanel todo, progress, done;
  public Board() {
    initWidget(uiBinder.createAndBindUi(this));
  }
}
```

Ajoutons cette classe directement dans le RootPanel.

Création des Post-it

De la même façon, créons un composant GWT composé d'un FocusPanel qui contient un TextBox car il n'est pas possible d'avoir une zone éditable qui soit aussi draggable. Créons maintenant la classe Task :

```
public class Task extends Composite {

  private static TaskUiBinder uiBinder = GWT.create(TaskUi
Binder.class);

  interface TaskUiBinder extends UiBinder<Widget, Task> {}
  @UiField TextBox text;
  @UiField FocusPanel panel;
  public Task(String text) {
    initWidget(uiBinder.createAndBindUi(this));
    text.setText(text);
    getElement().setDraggable(Element.DRAGGABLE_TRUE);
    text.getElement().setAttribute(«draggable», «false»);
  }

  String getValue() {
    return text.getText();
  }

  public HandlerRegistration addDragStartHandler(DragStart
Handler h) {
    return panel.addDragStartHandler(h);
  }

  public HandlerRegistration addDragEndHandler(DragEndHandler h) {
    return panel.addDragEndHandler(h);
  }

  public HandlerRegistration addValueChangeHandler(ValueChange
Handler<String> s) {
    return text.addValueChangeHandler(s);
  }
}
```

Ainsi que le fichier UiBinder correspondant :

```
<!DOCTYPE ui:UiBinder SYSTEM «http://dl.google.com/gwt/DTD/
xhtml.ent»>
<ui:UiBinder xmlns:ui=»urn:ui:com.google.gwt.uibinder»
  xmlns:g=»urn:import:com.google.gwt.user.client.ui»>
  <ui:style>
    .task {
      min-height: 50px;
      width: 178px;
      padding: 7px;
      background: #FC3;
      border: 1px solid #FD5;
      -moz-border-radius: 5px;
      -webkit-border-radius: 5px;
      border-radius: 5px;
      -moz-box-shadow: 3px 3px 5px rgba(0, 0, 0, .2);
      -webkit-box-shadow: 3px 3px 5px rgba(0, 0, 0, .2);
      box-shadow: 3px 3px 5px rgba(0, 0, 0, .2);
      cursor: move;
      display: inline-block;
      margin: 5px;
    }
    .content {
      margin-top: 15px;
      border: none;
      background: transparent;
      min-height: 35px;
      height: 100%;
      width: 100%;
    }
  </ui:style>
  <g:FocusPanel styleName=»{style.task}» ui:field=»panel»>
    <g:TextBox ui:field=»text» styleName=»{style.content}»>
</g:TextBox>
  </g:FocusPanel>
</ui:UiBinder>
```

Nous utilisons ici du CSS3 avec la propriété border-radius pour avoir un bord arrondi et box-shadow pour avoir une ombre.
Dans la classe Task, nous ajoutons les deux lignes suivantes :

```
getElement().setDraggable(Element.DRAGGABLE_TRUE);
text.getElement().setAttribute(«draggable», «false»);
```

Elles permettent d'indiquer au navigateur que notre élément doit être draggable mais que notre Textbox ne doit pas l'être.

Ajout des tâches sur le tableau

Lorsque l'on ajoute une tâche sur le tableau, on ajoutera des handlers sur les événements de drag (DragStart au début et DragEnd à la fin) permettant d'enregistrer l'élément en cours lorsqu'une opération de glisser/déposer est effectuée :

```
Task taskDragged = null;
private void newTask(final Task task, FlowPanel panel) {
```

```
 task.addDragStartHandler(new DragStartHandler() {
   @Override
   public void onDragStart(DragStartEvent event) {
     taskDragged = task;
   }
 });
 task.addDragEndHandler(new DragEndHandler() {
   @Override
   public void onDragEnd(DragEndEvent event) {
     taskDragged = null;
   }
 });
 panel.add(task);
}
```

Lorsqu'un « glisser » est initié, nous enregistrons la Task qui est en train d'être déplacée, que nous réinitialisons ensuite. Ajoutons maintenant un événement sur le click du bouton newTask permettant de créer une nouvelle tâche :

```
@UiHandler(«newTask»)
public void newTask(ClickEvent e) {
 newTask(new Task(«new task «), todo);
}
```

Déposer une tâche dans une colonne du tableau

Pour déposer une tâche dans une des colonnes du tableau, il faut écouter les événements de Drop :

- DragEnter : Lorsque la souris entre dans une zone droppable avec un objet qui est en train d'être glissé.
- DragOver : Lorsque la souris est au-dessus d'une zone droppable, il est nécessaire de définir cet événement et d'annuler les actions par défaut pour accepter que l'on puisse déposer un objet sur cette zone
- DragLeave : Lorsque la souris sort de la zone.
- Drop : Lorsque l'on lâche un objet sur la zone.

```
 @UiHandler({«todoContainer»,»progressContainer»,»doneContainer»,
»deleteTask»})
public void dragOver(DragOverEvent e) {
 e.preventDefault();
 ((Widget) e.getSource()).getElement().getStyle().setBackground
Color(«#ccc»);
}
@UiHandler({«todoContainer»,»progressContainer»,»doneContainer»,
«deleteTask»})
public void dragLeave(DragLeaveEvent e) {
 ((Widget) e.getSource()).getElement().getStyle().setBackground
Color(«transparent»);
}
@UiHandler(«todoContainer»)
public void dropTodo(DropEvent e) {
 acceptDrop(e, todo);
}

@UiHandler(«progressContainer»)
public void dropProgress(DropEvent e) {
 acceptDrop(e, progress);
}
```

```
@UiHandler(«doneContainer»)
public void dropDone(DropEvent e) {
 acceptDrop(e, done);
}
private void acceptDrop(DropEvent e, FlowPanel panel) {
 e.preventDefault();
 ((Widget) e.getSource()).getElement().getStyle().setBackground
Color(«transparent»);
 if (taskDragged != null) {
     panel.add(taskDragged);
 }
 taskDragged = null;
}
```

Lors d'un DragOver, nous annulons les actions par défaut du navigateur (e.preventDefault()) pour accepter le drop sur la zone et changeons la couleur de fond de la zone que nous réinitialisons lors du départ de zone (DragLeave) ou le drop. Lors du drop sur une zone, nous ajoutons la tâche dans le FlowPanel correspondant à la colonne où nous avons déposé notre Post-it. Faisons la même chose pour supprimer une tâche :

```
@UiHandler(«deleteTask»)
public void dropdeleteTask(DropEvent e) {
 e.preventDefault();
 ((Widget) e.getSource()).getElement().getStyle().setBackground
Color(«transparent»);
 if (taskDragged != null)
   taskDragged.removeFromParent();
 taskDragged = null;
}
```

Sauvegarde et chargement des tâches enregistrées

Utilisons maintenant le LocalStorage pour sauvegarder la liste de tâches et leurs positions. Le LocalStorage est une base de données de type clé/valeur qui est stockée sur le navigateur de l'utilisateur qui reste présente lors des visites ultérieures. Il permet d'avoir un cache, ou d'y stocker les préférences d'un site. Il n'a pas de limite de taille contrairement aux cookies qui sont aussi envoyés au serveur à chaque requête. Le SessionStorage fonctionne de la même manière, mais il est supprimé lorsque l'utilisateur ferme son navigateur. L'utilisation est simple en GWT :

```
Storage storage = Storage.getLocalStorageIfSupported();
storage.setItem(«todo», «Task1»);
storage.getItem(«todo»);
```

Enregistrons alors nos tâches dans une clé par colonne, chacune des tâches séparée par un pipe « | » :

```
Storage storage = Storage.getLocalStorageIfSupported();
private void loadTasks() {
 if (storage != null) {
   loadTasks(storage.getItem(«todo»), todo);
   loadTasks(storage.getItem(«progress»), progress);
   loadTasks(storage.getItem(«done»), done);
 }
}
```

```
private void saveTasks() {
 if (storage != null) {
   storage.setItem(«todo», saveTasks(todo));
   storage.setItem(«progress», saveTasks(progress));
   storage.setItem(«done», saveTasks(done));
 }
}
private String saveTasks(FlowPanel panel) {
 StringBuilder sb = new StringBuilder();
 for (int i = 0; i < panel.getWidgetCount(); i++) {
   Task task = (Task) panel.getWidget(i);
   if (task != null) {
     if (i > 0)
       sb.append('|');
     sb.append(task.getValue());
   }
 }
 return sb.toString();
}
private void loadTasks(String items, FlowPanel panel) {
 if (items != null && !items.isEmpty()) {
   String[] listItems = items.split(«[|]»);
   if (listItems != null && listItems.length > 0)
     for (String item : listItems) {
       if (item != null && !item.trim().isEmpty())
         newTask(new Task(item), panel);
     }
 }
}
```

Il est maintenant nécessaire d'appeler la méthode saveTasks() lorsqu'une action est faite :

- Lors de l'ajout d'une nouvelle tâche
- Lors de la suppression d'une tâche
- Lorsque l'on déplace une tâche
- Lorsque le texte d'une tâche a changé (s'abonner à l'événement ValueChangeEvent de l'objet Task)

Graphique d'avancement

La nouvelle balise Canvas est une zone de dessin qui permet en javascript (et donc en GWT) de dessiner avec une accélération matérielle (pour les derniers navigateurs). Plusieurs fonctionnalités sont disponibles pour dessiner des points, des droites, des figures géométriques, des arcs, colorier des zones, etc.

Créons un Canvas en indiquant la taille du Canvas et la taille en pixel du dessin (définissant la qualité du dessin) :

```
private void createCanvas() {
 canvas = Canvas.createIfSupported();
 if (canvas != null) {
   canvasContainer.setWidget(canvas);
   canvas.setCoordinateSpaceHeight(30);
   canvas.setCoordinateSpaceWidth(30);
   canvas.setPixelSize(30, 30);
   displayGraph();
 }
}
```

Il faut ajouter ce Canvas à notre écran en utilisant, par exemple, un SimplePanel placé à droite de notre titre (avec un float:right en css)
Nous allons réaliser un camembert montrant l'avancement de nos tâches (les tâches à faire en rouge, les tâches en cours en orange et les tâches finies en vert.
Pour cela, il faut compter le nombre de tâches totales par colonne et ensuite calculer l'angle de chaque zone en radian : un cercle entier à un angle en radian de 2PI. On peut ainsi calculer facilement l'angle de chaque zone (en additionnant avant l'angle précédent) :

```
private void displayGraph() {
 int width = canvas.getCoordinateSpaceWidth();
 int height = canvas.getCoordinateSpaceHeight();
 double radius = Math.min(width, height) / 2.0;
 double cx = width / 2.0;
 double cy = height / 2.0;
 Context2d context = canvas.getContext2d();
 context.clearRect(0, 0, width, height);
 double todoSize = todo.getWidgetCount();
 double progressSize = progress.getWidgetCount();
 double doneSize = done.getWidgetCount();
 double total = todoSize + progressSize + doneSize;
 double todoRad = Math.PI * 2.0 * (todoSize / total);
 double progressRad = todoRad +  Math.PI * 2.0 * (progressSize / total);
 double doneRad = progressRad + Math.PI * 2.0 * (doneSize / total);
 displayPie(context, radius, 0, todoRad, cx, cy, CssColor.make
(«red»));
 displayPie(context, radius, todoRad, progressRad, cx, cy, Css
Color.make(«orange»));
 displayPie(context, radius, progressRad, doneRad, cx, cy, Css
Color.make(«green»));
}
```

Enfin, créons le code pour dessiner un arc de cercle et le remplir :

```
private void displayPie(Context2d context, double radius,double
beginRad, double endRad,double cx, double cy, FillStrokeStyle color) {
 context.beginPath();
 context.moveTo(cx, cy);
 context.arc(cx, cy, radius, beginRad, endRad, false);
 context.lineTo(cx, cy);
 context.setFillStyle(color);
 context.fill();
}
```

Nous créons ici un nouveau crayon. Positionnons-le en centre du cercle pour ensuite tracer l'arc de cercle de l'angle voulu et revenir en centre pour fermer la zone et la remplir avec une couleur.
Finissons notre projet en ajoutant l'appel à la fonction displayGraph() chaque fois que l'on souhaite rafraîchir le Graph (ajouter, supprimer, déplacer une tâche).
Retrouvez le code source de ce projet sur google code : http://sfeir.googlecode.com/svn/trunk/PlanningBoard et la version en ligne http://planning- board.appspot.com

Patrice de **Saint Stéban**
Ingénieur d'étude et de développement - Sfeir

SQL Server « Denali » : Quoi de neuf, quelles perspectives ?

Denali est le nom de code de la prochaine version majeure de SQL Server. Cette nouvelle version apporte de nombreuses nouveautés non seulement à son moteur de base de données mais aussi aux services d'intégration et de gestion des données ou encore à la plateforme de Business Intelligence (BI).

Denali permet à l'IT de garder le contrôle tout en laissant les utilisateurs tirer le meilleur parti des données du système d'information. Pour y arriver, Denali est construit autour des trois axes suivants :

- Être la plateforme des applications "Mission Critical" avec un TCO faible.
- Ouvrir de nouvelles perspectives aux entités métier en généralisant l'accès aux données.
- Etre prêt pour le Cloud selon vos besoins.

Chacun de ces axes de développement est soutenu par plusieurs innovations apportées avec cette dernière version de SQL Server.

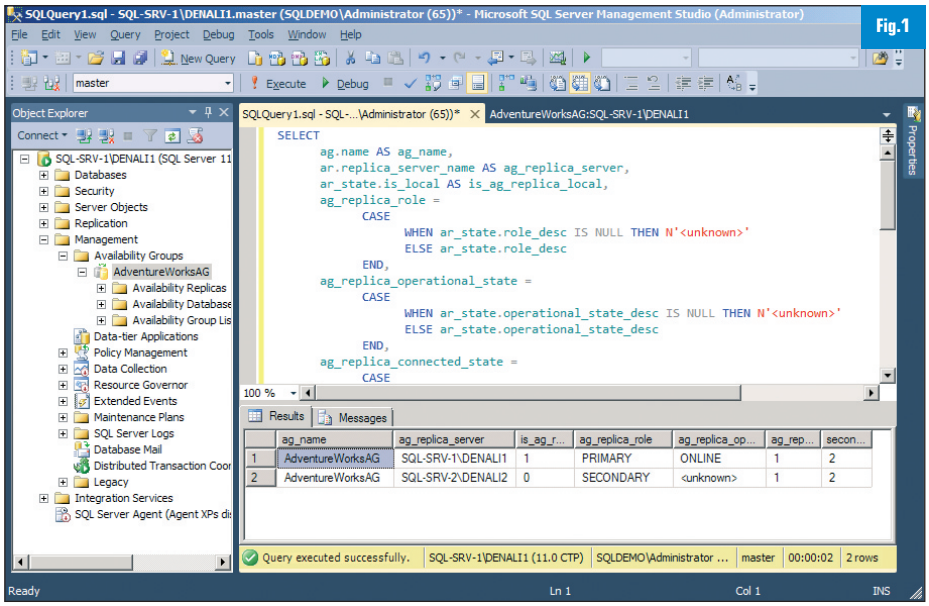


Fig.1

Plateforme "Mission Critical"

Pour fournir un service capable de répondre aux besoins des applications "Mission Critical" des entreprises, les nouveautés de Denali se trouvent dans la protection et la disponibilité des données, la performance du moteur et la capacité à respecter les besoins de conformité des entreprises.

La protection et la disponibilité des données ont été améliorées grâce à la combinaison des trois éléments que sont : **SQL Server AlwaysOn**, la réduction du nombre d'opérations nécessitant une maintenance de la base et le support de Windows Server Core.

AlwaysOn est une solution intégrée de haute disponibilité et de reprise en cas de désastre. Cette solution améliore la capacité à redonder le service au sein d'un datacenter ou sur plusieurs datacenters [Fig.1].

Pour mieux garantir la disponibilité des bases de données des applications critiques, AlwaysOn introduit une unité logique qui s'appelle "*Availability Group*". Cette unité permet de regrouper plusieurs bases de données et objets en dépendance et maintient cette cohérence lors d'une bascule. Les bases faisant partie d'un *Availability Group* sont mises en miroir entre différents nœuds du cluster spécifiés par l'administrateur. A la différence d'un miroir actuel, avec AlwaysOn il est possible d'avoir jusqu'à quatre réplicas d'une base, ce qui fait un total de cinq instances d'une base de données. Une seule copie de la base est accessible en lecture / écriture, c'est la base primaire, les autres sont des copies secondaires. Ces copies secondaires sont accessibles en lecture seule, ce qui permet de pouvoir faire des sauvegardes ou exécuter des rapports sans perturber le fonctionnement de la base primaire. L'utilisation de copie dans le fonctionnement d'un cluster AlwaysOn permet de s'affranchir de disques partagés. Il reste possible d'utiliser un cluster à bascule traditionnel avec une baie SAN, mais Denali supporte aussi des serveurs avec des disques en attachement direct.

En maintenant une copie des bases et en les groupant dans cette unité logique qu'est le groupe de disponibilité, il est possible de maintenir en fonctionnement une application qui utiliserait plusieurs bases de données sans surcoût d'administration et en réduisant le coût d'acquisition du matériel. Le nom virtuel utilisé lors de la création du groupe de disponibilité permet non seulement d'assurer une compatibilité avec des applications qui n'ont pas nécessairement été prévues pour fonctionner en cluster mais aussi une bascule extrêmement rapide. Avec cette nouvelle solution, le temps de bascule est de l'ordre d'une minute (par rapport à plusieurs minutes avec les solutions actuelles).

L'une des sources d'indisponibilité d'un système est le reboot suite à la mise à jour de correctifs de sécurité. Le support de Windows Server Core, la version la plus épurée de Windows actuellement disponible, installe moins de composants et réduit de 50 à 60% le nombre de reboots nécessaires suite à l'installation de correctifs de sécurité et permet donc d'améliorer la disponibilité globale du service rendu par SQL Server.

Denali introduit le **stockage en colonne** dans le moteur de stockage. Ce nouveau

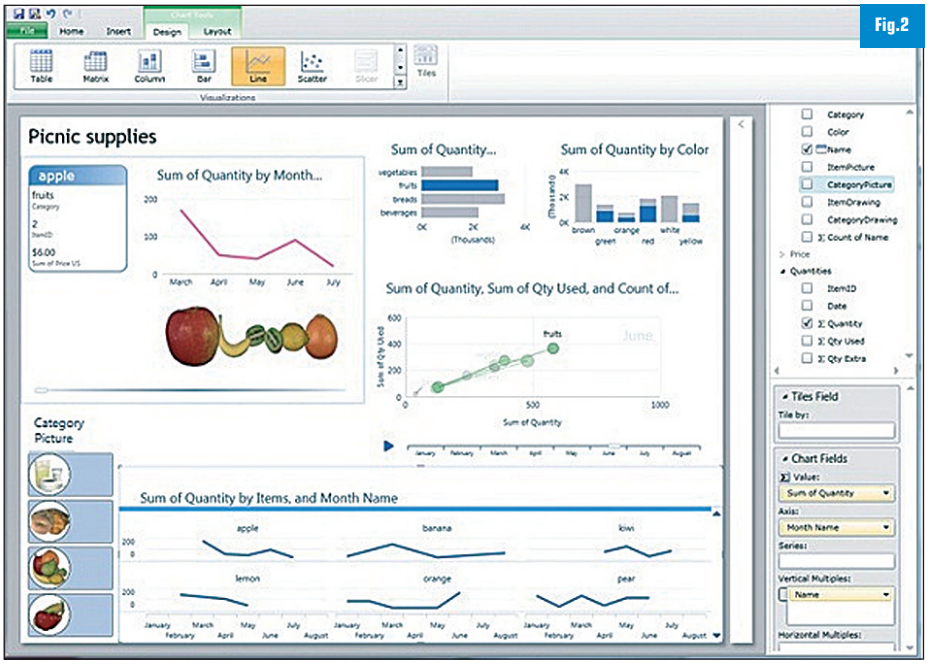


Fig.2

mode de stockage crée un index des valeurs de chaque colonne, la recherche dans la colonne est donc très rapide et permet d'améliorer jusqu'à un facteur de 10 les performances des requêtes sur des bases dont le schéma est en étoile. Par rapport à une base de données traditionnelle, le nombre d'I/O se trouve réduit pour les mêmes opérations.

Les performances de la recherche en texte intégral ont été nettement améliorées grâce à la réécriture complète du moteur. En stockant les documents dans une File Table (table spéciale dans laquelle on peut stocker des fichiers accessibles comme un répertoire depuis le système d'exploitation), Denali peut effectuer une requête sur 350 Millions de documents en moins de 3 secondes.

Pour terminer ce tour d'horizon des capacités de Denali à répondre aux attentes Mission Critical des entreprises, nous allons voir comment il répond aux besoins de sécurité et de conformité.

L'apport du **User-Defined Server Roles** permet à l'administrateur de créer des rôles d'administration et déléguer plus finement qu'avec 2008 R2 certaines opérations d'administration. L'utilisation de ces rôles personnalisés est très utile par exemple dans le cas où l'on souhaite déléguer à un administrateur uniquement la gestion des mots de passe d'un login SQL associé à une application.

Le moteur de base de données apporte aussi la notion de **Contained Database**, il est non seulement possible de séparer les rôles entre les administrateurs et les utilisateurs et donc de réduire la création de login inutiles mais aussi d'isoler complètement la base de données de son moteur. Les Contained Database contiennent aussi les plans de maintenance ou les procédures de sauvegarde rendant simples le déploiement et le déplacement d'applications.

Côté chiffrement des données, Denali implémente les recommandations d'algorithmes cryptographiques définis dans la NIST Suite-B. C'est-à-dire une exploitation complète des piles de chiffrement fournies par Windows 2008 :

- La génération de Hash avec les algorithmes SHA2_256 et SHA2_512.
- L'utilisation de AES (Advanced Encryption Standard) comme algorithme de chiffrement par défaut des clefs maître pour le service et pour de la base de données.
- Le support de RC4 uniquement pour des raisons de rétrocompatibilité mais ne peut plus être utilisé pour du chiffrement.

Que ce soit à travers les améliorations en termes de disponibilité des données, de performance du moteur de base de données ou de sécurité, Denali peut fournir la plateforme des applications "Mission Critical".

Nouvelles perspectives sur les données

Les investissements technologiques faits dans Denali sur la gestion des données ont pour but de permettre aux clients de faire face à l'explosion de données que l'on vit actuellement. Les nouvelles capacités de Denali permettront d'explorer ces données, qu'elles soient structurées ou non, ou provenant du Cloud. Pour permettre d'ouvrir de nouvelles perspectives pour les entreprises, il est nécessaire de rendre les données cohérentes et crédibles.

Plusieurs outils intégrés à Denali permettent d'effectuer ces tâches. Quand le volume de données devient trop important, Denali permettra de construire un Parallell Data Warehouse capable de monter en charge.

Avec le projet **Crescent**, Denali rend l'exploration des données accessible à tous. En particulier aux utilisateurs qui ont besoin d'accéder aux données métier mais qui n'ont pas nécessairement les compétences pour écrire des requêtes. Grâce à une interface web, graphique, simple, fortement interactive, proche du ruban Office, **Crescent** permet aux utilisateurs de véritablement naviguer dans les données, d'afficher des rapports ou de créer des animations en quelques clics. Cette interactivité permet, par exemple, d'identifier rapidement des tendances, des anomalies et de les présenter de façon explicite. Crescent repose sur un modèle de BI Sémantique complètement intégré à PowerPivot [Fig.2].

Le **Modèle de BI Sémantique** aussi appelé **BISM** (BI Semantic Model) est une nouveauté majeure apportée par Analysis Services de Denali, c'est lui qui fournit le modèle de données à toutes les interfaces utilisateur de BI comme Crescent, Excel, Reporting Services ou Sharepoint Insights.

L'architecture du modèle BISM peut être vue à travers trois couches :

- Le modèle de données qui est exposé aux applications clientes. BISM est un modèle hybride qui supporte la modélisation multidimensionnelle et tabulaire des données. Le développeur utilisera le même outil (BI Development Studio - BIDS) pour créer ces deux types de projets. Les applications comme Crescent utiliseront l'interface tabulaire alors que des applications riches comme Excel utiliseront l'interface multidimensionnelle.
- La couche intermédiaire est la Business Logic, utilisant des expressions DAX (Data Analysis eXpressions) ou MDX (MultiDimensional eXpressions). DAX est proche de ce que l'on peut connaître avec Excel et est relativement simple d'utilisation alors que MDX, certes plus puissant et plus souple, nécessite un apprentissage plus important.

- La dernière couche est la couche d'accès aux données. Les types d'accès aux données possibles sont soit en mode cache (MOLAP et Vertipaq) soit en mode temps réel (ROLAP et DirectQuery). Le mode MOLAP correspond au moteur de stockage utilisé par Analysis Services dans les anciennes versions. Le mode VertiPaq lit le contenu de la source de données et les met dans un stockage en colonne bénéficiant ainsi de compression et d'algorithmes de requête multithread. Le mode temps réel doit laisser ces bénéfices au profit de l'accès à une donnée à jour.

Avec la CTP3 de Denali [Fig.3] selon le choix du type de projet dans BI Development Studio (BIDS), à savoir soit le modèle tabulaire soit le modèle multidimensionnel, vous serez limité au choix du langage de modélisation associé ainsi qu'au type d'accès aux données. Un projet Tabulaire amènera le développeur de modèle à utiliser DAX et Vertipaq/DirectQuery, alors qu'un projet multidimensionnel l'amènera sur l'utilisation de MDX et MOLAP/ROLAP. Il est aussi important de noter qu'une instance SSAS n'implémente qu'un seul des deux modes, le choix est fait lors de l'installation.

PowerPivot et Analysis Services, bien que ciblant des utilisateurs différents utilisent tous les deux le nouveau modèle BISM. Dans le cas de PowerPivot, le modèle est stocké dans le classeur Excel alors que pour Analysis Services, le modèle est stocké sur le serveur. L'utilisation du même modèle de BI permet des transitions transparentes entre des applications BI personnelles et des applications BI d'équipe ou d'entreprise. Un classeur construit par un business user peut ainsi rapidement et facilement devenir une application Analysis Services.

L'exploitation des données par la BI ne prend toute sa valeur que si les données fournies sont cohérentes. Le **nettoyage et la cohérence des données** est un défi majeur que toute les entreprises doivent actuellement relever. Denali apporte nativement les outils permettant d'y répondre.

Master Data Services (MDS) se démocratisant avec l'arrivée d'un add-on à Excel, on laisse ainsi aux personnes qui connaissent les données la possibilité de les corriger. Les données peuvent être filtrées avant d'être chargées et modifiées dans Excel, seulement les modifications sont publiées dans la base. Avant de procéder à des ajouts dans MDS, MDS permet de confirmer l'absence d'enregistrements en double en comparant deux sources de données (par exemple les données de MDS avec un fichier Excel). L'add-in pour Excel sait exploiter DQS (Data Quality Services, voir ci-dessous) pour proposer des suggestions lors de la mise à jour de données. MDS est aussi accessible à travers une simple application web en Silverlight permettant facilement d'ajouter ou supprimer des membres ou même de les déplacer dans la hiérarchie.

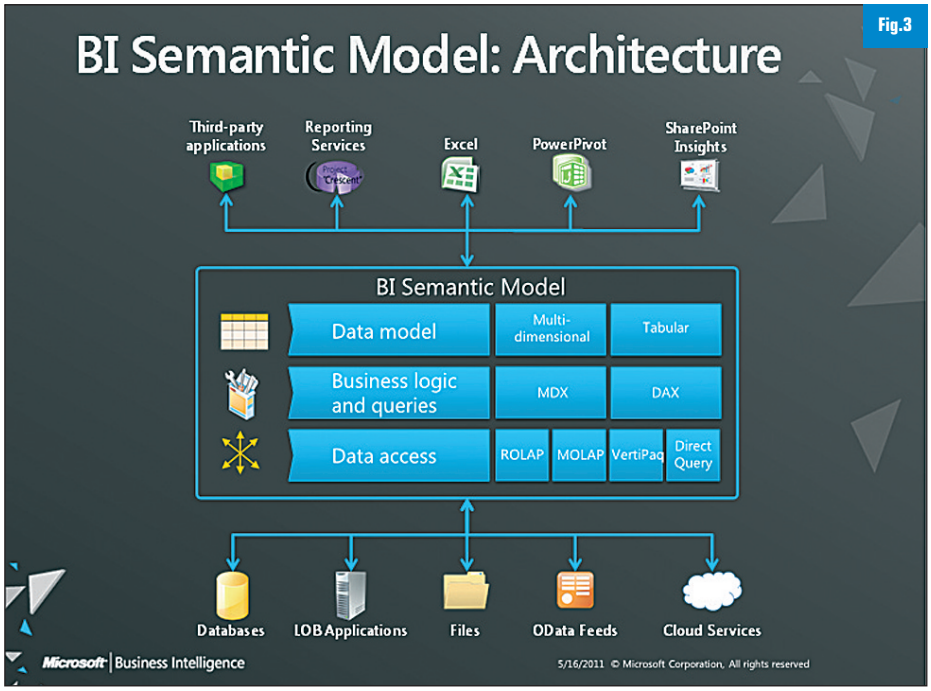


Fig.3

Data Quality Services (DQS) est une nouvelle capacité apportée par Denali à SQL Server. Ce service permet de créer et maintenir une base de connaissances de la qualité des données. En capturant ainsi la connaissance de l'organisation, les clients peuvent mieux nettoyer, profiler, associer leurs données et au final améliorer leur qualité. Les clients implémentant DSQ peuvent aussi consommer des sources de données tierces sur le Windows Azure Data Marketplace pour valider ou nettoyer leurs données.

SQL Server Integration Services (SSIS) sait utiliser MDS et DQS pour s'assurer que les données chargées dans le datawarehouse sont correctes quand les utilisateurs vont commencer à les analyser.

Denali apporte les outils et services nécessaires pour nettoyer et rendre les données cohérentes pour être analysées par les business user. Le nouveau modèle de BI Sémantique (BISM) exploitera ces données selon un modèle tabulaire ou multidimensionnel selon le type de client amené à se connecter sur le modèle de données créé. Parmi les différents clients possibles, on retrouve le projet Crescent permettant à tout utilisateur d'explorer les données selon un modèle tabulaire ou Excel avec PowerPivot. L'utilisation de BISM permet de transformer simplement une application BI d'utilisateur en une application disponible pour l'ensemble de l'organisation.

Etre prêt pour le cloud selon vos besoins

La prochaine version de SQL Server a été conçue dès le début pour répondre aux besoins des entreprises en termes de flexibilité et adresser les défis du cloud privé et public. Pour répondre à ces contraintes, SQL Server repose sur la même architecture pour les déploiements on-premises, les Appliance ou les approches cloud. Cette approche permet non seulement d'uniformiser les outils d'administration et de développement mais aussi d'assurer la continuité des applications entre les différents déploiements. Il est donc possible de commencer un développement pour une base de données virtualisée, puis de migrer vers le cloud public dans un second temps sans avoir à modifier l'application.

Le projet **Juneau** est le nom de code pour les outils de développement de SQL Server (SSDT). L'approche prise est de fournir un environnement intégré à Visual Studio pour les développeurs de base de données. Le Server Explorer a été retravaillé pour inté-

grer des fonctionnalités jusque-là réservées à SSMS et simplifier ainsi la création ou l'édition de bases de données. Par exemple lors de la suppression d'une base de données, Juneau affichera les implications liées à l'opération. Juneau permet au développeur de travailler sur les modèles de bases de données en étant connecté ou déconnecté de la base de données. Ce dernier mode permet au développeur de faire des tests en local avant de publier les modifications dans la base de données qui peut être une base sur site ou une base SQL Azure.

Juneau permet aussi et surtout de maintenir synchronisées les applications et les bases de données. C'est en utilisant un Entity Data Model (EDM) que se crée une référence entre le projet applicatif et le projet de base de données. Les changements effectués dans l'EDM sont propagés automatiquement vers le projet de base de données de façon incrémentale et inversement.

Pour compléter cette uniformité du développement applicatif, Denali apporte une symétrie entre les solutions on-premise et off-premise. Il est ainsi possible de déplacer les DAC (Data-Tier Application Component) de façon transparente entre les serveurs, le cloud privé ou SQL Azure.

La prochaine version de SQL Server, nom de code DENALI intègre plusieurs nouveautés dont **AlwaysOn** capable de délivrer une solution répondant aux besoins des applications les plus critiques des entreprises. Avec les services comme DQS (Data Quality Services), MDS (Master Data Services) et SSIS (SQL Server Integration Services), SQL Server permet de **nettoyer et assurer la qualité des données** fournies au datawarehouse.

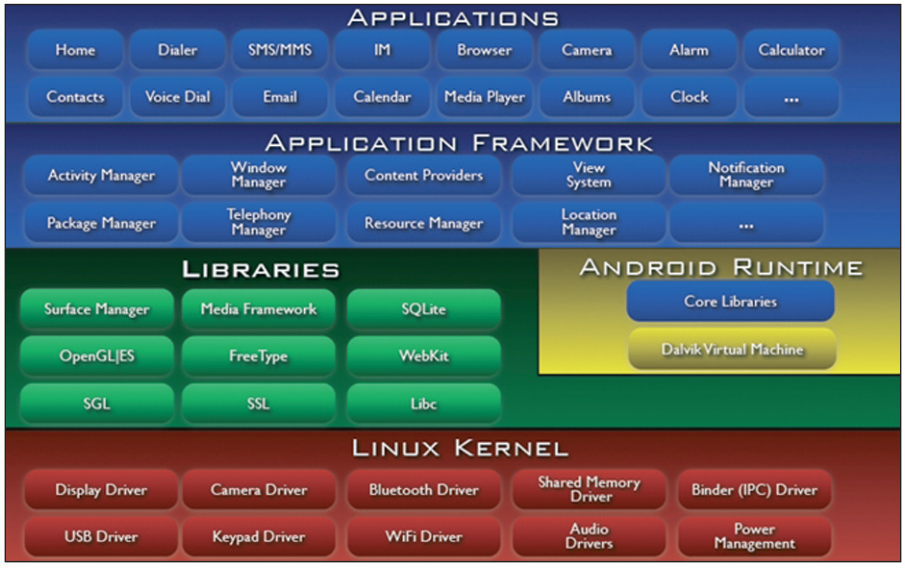
En procurant un nouveau modèle de BI sémantique (**BISM** – BI Semantic Model), Denali permettra non seulement à tout employé d'accéder à la BI grâce au projet **Crescent** mais aussi offrira une transition simple d'une simple application utilisateur vers la BI d'entreprise. Le travail du développeur avec le projet **Juneau** sera aussi simplifié en maintenant synchronisé le modèle d'entité de données entre le projet applicatif et le projet de base de données. Enfin, SQL Denali est une solution prête pour le Cloud en permettant le déplacement transparent des bases d'un serveur on-premise vers SQL Azure.

■ Damien **Caro**
Architecte infrastructure, Microsoft France

Devenez un développeur android

Android et son architecture

Avant de procéder à tout développement, il est primordial de connaître son architecture. A travers cet article, nous allons découvrir les entrailles du système et en comprendre son fonctionnement.



Android en quelques mots

Android (du nom de l'entreprise créatrice) est une plateforme dédiée au monde mobile comprenant

- Un système d'exploitation
- Des bibliothèques écrites en C/C++
- Des applications (Messagerie, calendrier…)

L'architecture Android se présente sous forme de couches logicielles comme le décrit la [Fig.1].

Linux le cœur du système

La couche la plus basse de l'architecture repose sur un noyau linux 2.6. Cependant, il ne s'agit pas d'un Linux proprement dit car X-windows n'est pas implémenté (gestion de fenêtrage) tout comme la librairie glibc qui n'est pas supportée (Android utilise une bibliothèque libc customisée, nommée Bionic)

Le choix d'un noyau Linux a été réalisé pour sa stabilité et sa performance, pour son modèle de sécurité, pour ses capacités d'abstraction avec le matériel et enfin pour son aspect open source et communautaire fort. A noter enfin que le noyau Linux a été patché offrant ainsi de nouvelles fonctionnalités. Le tableau ci-dessous liste quelques exemples de patchs

Patch	Fonctionnalité
Alarm	Mise en place de timers permettant par exemple de réveiller l'appareil quand il est en veille
Ashmem	Permet le partage de la mémoire entre applications
Binder	Permet la communication entre processus
Logger	Offre un système de logs
Wakelogs	Permet la gestion de l'alimentation

Les librairies

Ecrites en C/C++, les librairies sont utilisées par les composants du système Android et sont utilisables par le développeur via le framework Android :

- Surface Manager gère l'accès au sous-système d'affichage
- Media Framework permet la gestion de fichier multimédia (Audio et vidéo)
- SQLite, le moteur de base de données relationnel
- OpenGL, la bibliothèque graphique 3D
- FreeType gère les bitmap et le rendu des polices
- WebKit permet la navigation internet
- SGL, le moteur graphique 2D
- SSL gère le protocole SSL
- Libc gère la librairie C

Entre le noyau Linux et les librairies il y a…

Entre le noyau et les librairies précédemment décrites, il existe la couche d'abs-

traction matérielle (Hardware Abstraction Layer : HAL). Cette couche fournit les interfaces devant être implémentées par le noyau. Dans cette couche, nous trouvons donc une interface permettant de gérer la caméra implémentée par le Diver Camera du noyau, une autre pour le bluetooth, etc. HAL a pour fonction première de séparer la plateforme logique des interfaces matérielles et de faciliter le portage des librairies sur différents matériels.

Le moteur d'exécution Linux (Android Runtime)

Chargé d'exécuter les applications, le moteur Android comporte deux éléments :

- Des bibliothèques de base
- La machine virtuelle Dalvik

Les bibliothèques de base fournissent les fonctionnalités du langage JAVA (langage utilisé pour le développement d'applications Android) et des bibliothèques spécifiques à Android.

En lieu et place d'utiliser une JVM (Java Virtual Machine) classique, Android dispose de sa propre machine virtuelle nommée Dalvik. Basée sur une architecture de registre, Dalvik exécute les applications Android une fois celles-ci compilées au format requis (.dex) à l'aide de l'outil dx (Un fichier java étant compilé en .class, ce fichier étant ensuite compilé en .dex)

Il est important de noter que chaque application Android s'exécute dans son propre processus ayant sa propre instance de Dalvik car la VM Android a été écrite à des fins de multiples instanciations sans perte de performances. Enfin, Dalvik s'appuie sur le noyau Linux pour la réalisation des tâches dites de bas niveau telles que le threading, la gestion des processus et de la mémoire.

Le framework de développement

Cette couche intéresse tout particulièrement les développeurs car elle leur permet de créer des applications à l'aide d'une plateforme ouverte.

Utilisant le langage JAVA, le framework met à disposition un ensemble de classes utiles à la création d'applications sans oublier la mise à disposition de classes et méthodes permettant l'accès au matériel, à la gestion de l'interface graphique et aux ressources de l'application.

Enfin, notons que le framework Android se compose des services applicatifs suivants

- Activity Manager gère le cycle de vie des Activités (Activities)
- Views permet de créer les interfaces graphiques
- Notification Manager fournit un mécanisme d'envoi de messages aux utilisateurs
- Content Providers permet aux applications de partager des données entre elles
- Resource Manager offre l'externalisation de ressources telles que les chaînes de caractères pour la notion d'internationalisation, de style, de menus…

La couche applicative

Couche la plus haute de l'architecture, celle-ci contient l'ensemble des applications natives, tierces ou développées présentes sur l'appareil.

Toutes les applications présentes dans cette couche sont, comme nous l'avons vu précédemment, exécutées par le moteur d'exécution Android.

Nous venons de parcourir l'ensemble des couches de l'architecture Android, et d'avoir ainsi une vue d'ensemble de son fonctionnement. Avec ces notions en tête, il nous est donc plus facile d'aborder le développement d'applications Android. Néanmoins pour réaliser cette tâche, nous devons disposer d'un environnement de développement correctement configuré, c'est ce que nous allons aborder à présent.

L'INSTALLATION DU FRAMEWORK DE DÉVELOPPEMENT

Les outils.

Pour développer des applications Android, nous avons besoin d'un IDE tel qu'Eclipse (www .eclipse.org). Nul besoin d'une version spécifique, la version classique suffit amplement.

Une fois l'IDE téléchargé et installé, nous devons le configurer pour pouvoir utiliser le SDK d'Android afin de créer notre première application.

La première étape consiste à télécharger le SDK Android disponible à cette adresse : http://developer.android.com/sdk/index.html

A l'heure où cet article est écrit, la dernière version d'Android disponible est l'Android-r12 (A noter que la version archivée est à préférer)

Chacun a ses préférences d'installation, personnellement j'ai choisi l'option de créer un répertoire ANDROID à la racine de mon disque dur où j'ai placé le répertoire d'Eclipse (C:\ANDROID\Eclipse) et celui du SDK Android (C:\ANDROID\android-sdk-windows)

Installation du SDK

La décompression de l'archive du SDK Android ne suffit pas, il faut maintenant installer des packages supplémentaires nécessaires au développement d'application. Cette tâche se réalise en exécutant le

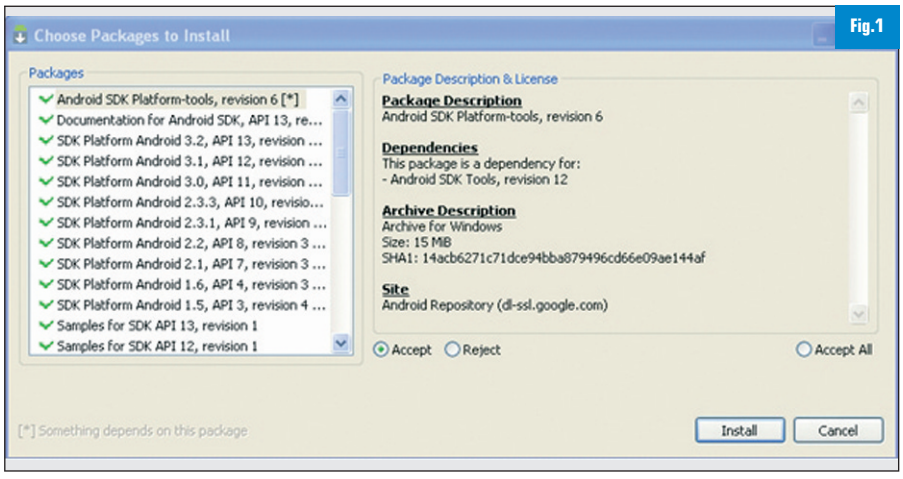


Fig.1

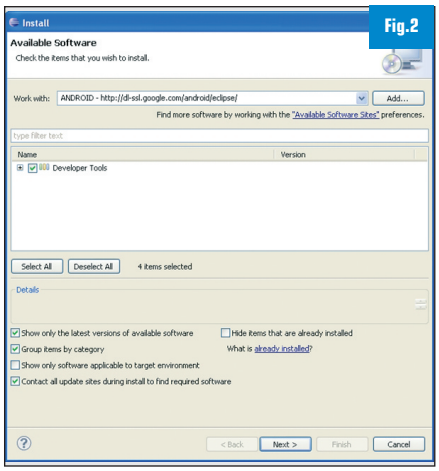


Fig.2

VOTRE CODE EST MULTI-PLATEFORMES
Windows, Linux, .Net, Java, PHP, Mac, J2EE, XML, Internet, SaaS, Cloud, Windows Phone, CE, Mobile, iOS, Android, ...

ATELIER DE GÉNIE LOGICIEL PROFESSIONNEL

WINDEV MOBILE 17

NOUVELLE VERSION

DÉVELOPPEZ 10 FOIS PLUS VITE

917 NOUVEAUTÉS

ET MAINTENANT SUR iPHONE & iPAD

VERSION EXPRESS GRATUITE
Téléchargez-la !

Développez avec la puissance et la facilité légendaire de WINDEV

iOS (iPHONE...)
ANDROID
WINDOWS PHONE
WINDOWS MOBILE

DEMANDEZ LE DOSSIER GRATUIT
Dossier gratuit 260 pages sur simple demande. Tél: **04.67.032.032** info@pcsoft.fr

Fournisseur Officiel de la Préparation Olympique
www.pcsoft.fr

SDK Manager se trouvant à la racine du SDK Android.
Lors de la première exécution du SDK Manager, un ensemble de packages nous sont proposés pour installation. Nous les acceptons tous (Accept All) et poursuivons en cliquant sur le bouton « Install » [Fig.1].

Interfacer Eclipse et le SDK

Pour développer des applications Android avec Eclipse, nous avons besoin du plugin ADT permettant de réaliser l'interface entre le SDK Android et Eclipse. Pour cela, il est nécessaire d'ouvrir Eclipse.
La première étape est de créer un espace de travail (WorkSpace), se caractérisant par un répertoire sur votre disque dur (C:\WKP_ANDROID pour ma part).
Une fois cette étape réalisée, nous pouvons procéder à l'installation du plugin ADT via le menu « help » d'Eclipse puis par le choix de l'option « install new software ».
Dans la fenêtre qui se présente à nous, cliquer sur le bouton ADD puis saisir les paramètres suivants :

- Name: ANDROID
- Location : http://dl-ssl.google.com/android/eclipse/

Après avoir cliqué sur le bouton OK, une recherche internet automatique est effectuée afin de lister les packages disponibles.
Cocher l'option « Developer tools » puis cliquer sur le bouton « next » afin de pouvoir procéder à l'installation de toutes les applications [Fig.2].

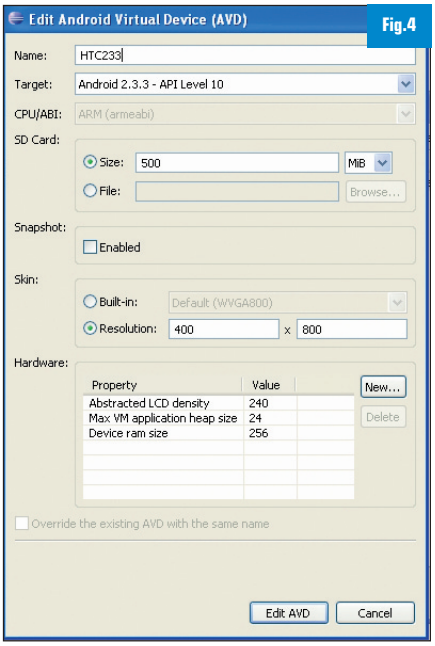


Fig.4

Spécification du SDK Android

Nous devons à présent spécifier le répertoire du SDK Android dans Eclipse afin de pouvoir développer des applications avec cet IDE.
Dans le menu windows / préférences choisir le menu Android puis dans la zone de texte « SDK Location » préciser le chemin du SDK Android (C:\ANDROID\android-sdk-windows). Une fois ceci réalisé, nous devons voir apparaître l'ensemble des SDK Android disponibles (1.5, 1.6.... 3.2) [Fig.3]

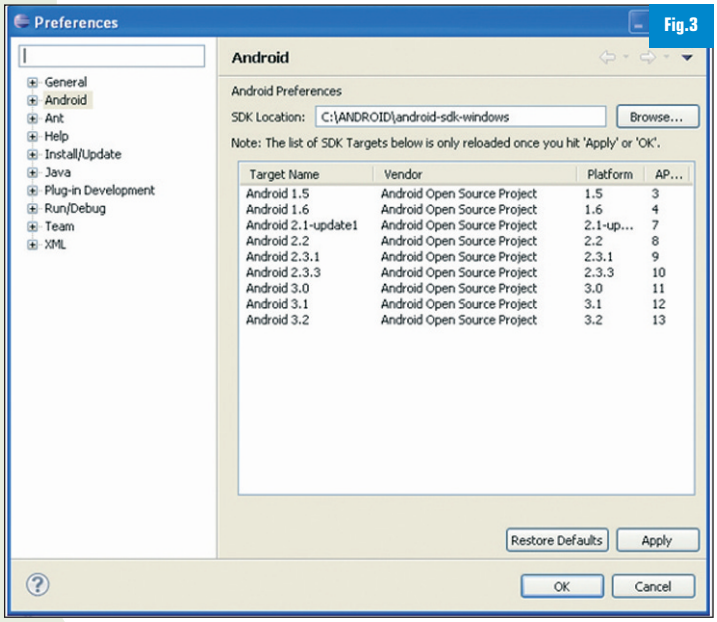


Fig.3

Création d'un device virtuel

Notre environnement étant à présent correctement configuré, nous pouvons développer notre première application. Cependant, pour tester notre code nous aurons besoin d'un device Android. Un device peut être physique (connecté à l'ordinateur via USB) ou émulé. Pour la seconde option, nous avons besoin de créer un device virtuel via le menu Windows / SDK et AVD Manager. Dans l'interface de création, cliquer sur le bouton « New » puis saisir les paramètres correspondant au device à émuler [Fig.4]. Une attention particulière est à apporter sur la version cible d'Android à utiliser (Target). En effet, à partir de la version 3, il est nécessaire de disposer d'une machine puissante et il n'est pas garanti que le device émulé s'exécute correctement.

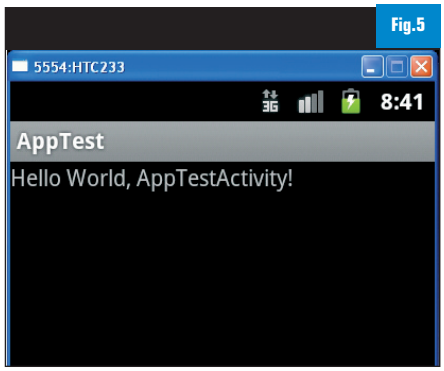


Fig.5

Création d'une première application

Nous disposons à présent d'un environnement Android tout à fait fonctionnel.
Pour vous en rendre compte, je vous invite à créer un nouveau projet Android (File>New >Android Projet) avec les paramètres suivants :

Paramètre	Valeur
Project Name	AppTest
Build Target	Android 2.3.3
Application name	AppTest
Package name	com.apptest
Activity	AppTestActivity
Min SDK	10 : Correspondant à Android 2.3.3

Puis de l'exécuter via le device émulé précédemment (Run>Run as > Android application). A noter qu'il n'y pas de code à saisir car l'application dispose de tous les éléments nécessaires et affiche par défaut le texte « Hello World, AppTestActivity » [Fig.5].
L'installation du SDK Android est à présent terminée. Cependant des packages complémentaires (Third party add-ons) et des mises à jours du SDK peuvent êtres installées grâce à SDK Manager et son menu Availables packages. Je vous invite donc à utiliser cet outil de façon régulière.
Libre à vous maintenant de mettre à profit vos nouvelles connaissances sur l'architecture Android et votre imagination afin de publier prochainement votre application sur l'Android Market.

Aurélien **Vannieuwenhuyze**

Architecte Flex Mobilité

http://aurelien-vannieuwenhuyze.com

Développer et publier sa première application Android

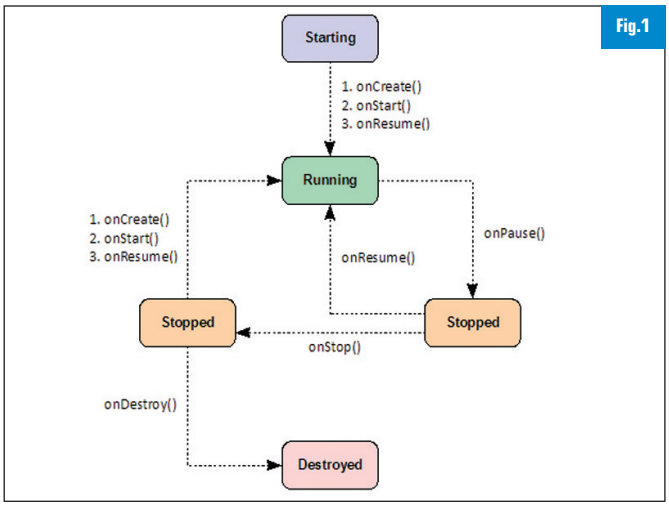
Au sein d'un monde de la mobilité promis à un bel avenir et en pleine effervescence, la plateforme Android s'est installée durablement comme premier système d'exploitation pour les smartphones. De fait, de plus en plus de développeurs se dirigent vers la plateforme pour le développement d'applications mobiles. De nombreuses ressources existent permettant de se focaliser sur des points précis du SDK Android mais aucune ressource ne donne une vision globale de la réalisation d'une application Android, de sa conception à sa publication. Cet article a pour but de combler ce manque en détaillant la réalisation d'une application Android de A à Z.

La plateforme Android accueille chaque jour de plus en plus de développeurs attirés par son côté ouvert, sa prédominance sur le marché des smartphones mais également par la qualité de sa documentation. Si la réalisation d'une première application de type "Hello World" se révèle aisée, le développeur éprouve ensuite plus de difficultés à réaliser une véritable application en vue d'une publication. La faute à la complexité inhérente aux immenses possibilités du framework Android.

Partant de ce constat, cet article se propose de détailler les différentes étapes dans la réalisation d'une application Android réelle qui sera publiée sur le market Android de Google. Le but de cet article étant avant tout de présenter la démarche de réalisation d'une application Android de A à Z, nous partirons du principe que le lecteur est déjà familiarisé avec les notions de base du framework et qu'il a déjà un environnement Eclipse configuré avec le plugin Android Development Tools (ADT). Pour notre application, il est important d'avoir assimilé le modèle de composants Android avec en point d'orgue les Activity et les Intents. L'OS Android manage ces composants et il est bon de rappeler le cycle de vie des activités [Fig.1] afin d'éviter certains écueils lors du développement.

Pour permettre à une application d'interagir avec le cycle de vie d'une activité, il suffit d'overrider les méthodes telles que *onCreate*, *onResume* ou *onStop* au sein de sa classe dérivant de Activity.

Fig.1



Enfin, les IHM de l'application seront définies au sein de fichiers XML afin de séparer leur définition de leurs traitements.

Cahier des charges

La définition du cahier des charges de notre application permet de préciser clairement ce qu'elle doit faire. Basiquement, elle doit permettre d'afficher la météo de la ville actuelle de l'utilisateur sur plusieurs jours. Fonctionnellement cela reste simple mais l'intérêt est avant tout de proposer une certaine richesse technique au niveau des API qu'elle utilisera. A ce titre, la ville actuelle de l'utilisateur sera récupérée grâce au service de localisation du téléphone et utilisera le GPS si ce dernier en est équipé. En outre, depuis l'écran d'accueil affichant la météo de la ville actuelle, un menu donnera accès à une seconde activité permettant à l'utilisateur de saisir une liste de villes dont il souhaite connaître la météo actuelle. L'ajout d'une nouvelle ville à cette liste se faisant également via un menu dédié. Enfin, le clic simple sur un élément de la liste permettra d'accéder à l'écran détaillé de la météo de cette ville sur plusieurs jours. Quant au clic long sur un élément, il devra permettre le retrait de la ville de la liste de suivi.

Spécifications détaillées

Il est intéressant de prototyper les écrans IHM de l'application durant la phase de spécifications détaillées afin d'avoir un meilleur aperçu de ce que sera l'application et son ergonomie. Pour faire ce prototypage efficacement, nous allons utiliser l'application *Pencil* téléchargeable ici : http://code.google.com/p/evoluspencil/. Gratuite, elle propose une version standalone basée sur le moteur XULRunner et présente l'avantage d'être extensible via son système de collections de formes graphiques. Ainsi, pour des maquettes Android, on pourra utiliser des collections de formes adaptées présentes sur le site http://code.google.com/p/android-ui-utils/. Ces collections spécifiques Android installées, nous spécifions les 2 écrans de notre application météo [Fig.2].

Ces maquettes confirment l'axe de simplicité graphique choisi pour l'application. L'écran d'accueil affiche la météo de la ville actuelle de l'utilisateur ainsi que le détail des informations météorologiques courantes avec en particulier la température, l'humidité, les conditions de vent, les conditions générales ainsi qu'une

icône illustrant ces dernières. De plus, un bandeau sur la gauche de l'écran donne les prévisions météo pour les jours à venir.
Le menu de cet écran offre les 4 possibilités suivantes :

- Rafraîchissement de la ville actuelle via utilisation du service de localisation de l'OS
- Retour à la ville favorite de l'utilisateur, ce qui s'avère utile pour que l'utilisateur puisse revenir à sa ville actuelle lorsqu'il a consulté la météo d'une ville de la liste de suivi
- Affichage de l'écran de suivi de météo de villes
- Accès à l'écran de préférences en vue de définir une ville favorite, ce qui est utile en cas de problèmes avec le service de localisation de l'OS

Le deuxième écran présenté à la [Fig.2] affiche la liste des villes suivies par l'utilisateur avec pour chacune d'elles les conditions météo actuelles. L'ajout de ville se faisant via le menu "Ajouter ville".

Conception

Le service de localisation renvoie une position sous la forme d'un couple de coordonnées latitude / longitude. De fait, il est nécessaire au niveau de l'application de mettre en correspondance ce couple avec une adresse et plus précisément une ville. Pour mettre en place ce mécanisme de correspondance, l'objet Geocoder présent en natif au sein du framework Android, mais nécessitant une API tierce de localisation au runtime telle que GoogleMaps, semble être tout indiqué. Cependant, après quelques recherches il s'avère qu'il est connu pour un bug empêchant son fonctionnement sur certaines versions d'Android. De fait, nous devons trouver une solution alternative et pour cela nous nous orientons vers Google qui propose une API de Geocoding offrant un service de reverse geocoding qui est détaillé ici : http://code.google.com/intl/fr/apis/maps/documentation/geocoding/#ReverseGeocoding. Cette API propose un web service REST au format JSON qui renvoie les adresses associées à un couple de coordonnées latitude / longitude fourni en entrée. Le côté REST est un très bon point puisque ce type de web services est supporté nativement par Android, ce qui nous évitera d'avoir à alourdir notre application avec une bibliothèque tierce devant gérer du SOAP par exemple. La seule contrainte de ce service concerne sa disponibilité puisqu'il est limité à 2500 requêtes gratuites par jour. Néanmoins, cette limite est amplement suffisante pour notre application météo.

Maquettes IHM des écrans de l'application

Choix d'une API météo

Au niveau du service de météo, après avoir étudié les possibilités de l'API Yahoo Weather nous optons finalement pour une API de chez Google. En effet, même si l'API Yahoo est complète et bien documentée, le résultat qu'elle produit est sûrement un peu trop complexe et surtout trop détaillé pour les besoins de notre application météo. Cette API n'est pas réellement mise en avant par Google puisqu'on ne trouve quasiment aucune documentation sur son utilisation. Plutôt mise à disposition des développeurs de Gadgets Google, elle remplit parfaitement les besoins de notre application puisqu'à partir du nom d'une ville on pourra récupérer en retour au format XML la météo actuelle ainsi que les prévisions pour les jours à venir. Une requête sur ce web service REST aura la forme suivante : http://www.google.com/ig/api?weather=Marseille.

Utilisation du cache applicatif

L'application doit stocker le nom de la ville favorite de l'utilisateur, la liste des villes suivies ainsi que la dernière position connue en cas de problèmes avec le service de localisation. Le framework Android propose 3 types de stockage :

- Base de données embarquée dans le téléphone
- Ecriture au sein de fichiers
- API de gestion des préférences d'une application

Pour l'application météo, les deux dernières possibilités seront mises en œuvre. Le stockage de la ville favorite utilisera les préférences de l'application. Les autres informations nécessaires seront stockées au sein de fichiers dans la zone de cache de l'application. Il est bon toutefois d'avoir à l'esprit qu'à tout moment le contenu de ce cache pourra être libéré soit par l'utilisateur soit par l'OS si l'espace mémoire disponible venait à manquer.

Appels web services asynchrones

Pour garantir un confort d'utilisation optimal de l'application, les appels web services devront être réalisés de manière asynchrone pour ne pas bloquer l'IHM pendant leur exécution. Pour cela, ils seront effectués au sein d'une classe spécifique héritant de AsyncTask du framework Android. Cette classe permet d'implémenter une tâche exécutée par l'OS dans un thread séparé de celui de l'application appelante. La séquence d'affichage de la météo au démarrage de l'application est présentée au sein du digramme de la [Fig.3]. Ce diagramme montre bien comment l'activité principale de notre application est managée par l'OS Android. Ainsi, on reconnaît les appels aux méthodes spécifiques des activités permettant d'avancer au sein de leur cycle de vie. Une fois la position de l'utilisateur récupérée via l'appel à la méthode *getLastKnownLocation* de l'objet LocationManager, nous déléguons au sein de la tâche MeteoHomeLoader les appels web services qui vont permettre de récupérer in fine la météo de la ville de l'utilisateur modélisée au sein d'un objet métier Meteo. Cette délégation permet un affichage instantané du premier écran de l'application sans avoir à attendre la fin des appels web services comme cela aurait été le cas avec des appels synchrones.

Développement

Les choix techniques faits durant la conception, le développement de l'application météo peut enfin être attaqué. Pour démarrer, nous mettons en place les méthodes appelant les web services récupérant respectivement la ville associée à un couple de coordonnées latitude / longitude et la météo pour une ville donnée. La récupération de la ville, réalisée au sein de la méthode statique *getCityFromLocation* de la classe *CityGrabber* est détaillée ci-dessous :

```
public static String getCityFromLocation(Context context,
Location location) throws Exception{
    String city = null;
    DefaultHttpClient client = new DefaultHttpClient();
    List<NameValuePair> qparams = new ArrayList<NameValuePair>();
    qparams.add(new BasicNameValuePair("latlng", location
    .getLatitude() + "," + location.getLongitude()));
    qparams.add(new BasicNameValuePair("sensor", "true"));
    URI uri = URIUtils.createURI("http", "maps.googleapis.com", -1,
    "/maps/api/geocode/json",
    URLEncodedUtils.format(qparams, "UTF-8"), null);
    HttpGet getMethod = new HttpGet(uri);
    ResponseHandler<String> responseHandler = new BasicResponse
Handler();
    String responseBody = client.execute(getMethod, response
Handler);
    JSONObject object = new JSONObject(responseBody);
    String status = object.getString("status");

    if ("OK".equals(status)) {
      JSONArray addresses = object.getJSONArray("results").get
JSONObject(0)
    .getJSONArray("address_components");

       for (int i = 0; i < addresses.length() && city == null; i++) {
          JSONObject address = addresses.getJSONObject(i);
          JSONArray types = address.getJSONArray("types");

          if (types != null && "locality".equals(types.get(0))) {
              city = address.getString("short_name");
           }
       }

       Util.storeLastKnownCity(context, city);
    } else {
       city = Util.getLastKnownCity(context);
    }

    return city;
}
```

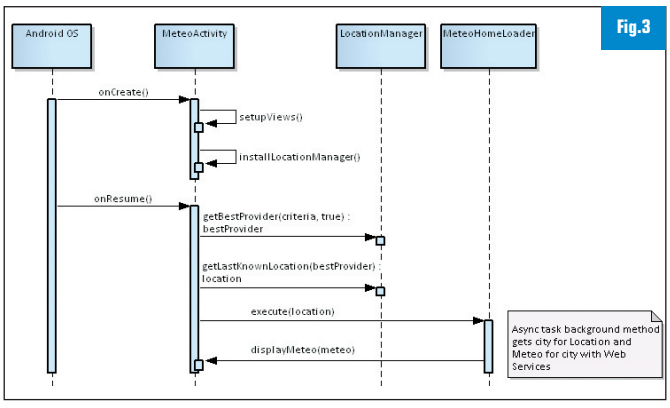


Fig.3

Séquence d'affichage de la météo

La consommation de web services REST est très simple grâce à l'utilisation de l'API HttpClient d'Apache embarquée en standard. Le traitement du résultat JSON l'est tout autant grâce à la présence d'une API bas niveau permettant sa manipulation. La ville actuelle obtenue, la récupération de la météo se fait de manière similaire au sein de la classe MeteoGrabber via la méthode statique *getMeteoFor*. La différence résidant dans le traitement du résultat du web service puisque dans ce cas on obtient un retour au format XML parcouru grâce aux classes de l'API DOM du framework.

Stockage de la ville

Afin de prévenir d'éventuelles indisponibilités du web service de correspondance entre coordonnées GPS et ville, la dernière ville retournée par ce service est stockée au sein du cache de l'application. Ce stockage est implémenté dans la méthode statique storeLastKnownCity de la classe de méthodes utilitaires Util :

```
public static void storeLastKnownCity(Context context, String
city) {
    if (context != null  &&  city != null) {
       File file = new File(context.getCacheDir(), LAST_KNOWN
_CITY_FILE_NAME);
       try {
          FileOutputStream fos = new FileOutputStream(file,
false);
          fos.write((city + NEW_LINE).getBytes());
          fos.close();
       } catch (Exception exc) {
          Log.d(LOG_TAG, exc.getMessage());
       }
    }
}
```

Chaque application possédant sa propre zone de cache, la récupération du chemin vers le répertoire de cache se fait via le contexte de l'application passé en entrée de méthode.

Traitements lourds non bloquants

Au lancement de l'activité principale de l'application météo, Android crée un thread principal nommée UI Thread chargé de gérer cette activité. Les appels de méthodes réalisés suite aux actions utilisateurs s'effectuent de fait au sein de l'UI Thread, ce qui pour des traitements longs entraîne le bloquage de l'IHM. Pour éviter ce bloquage détériorant l'expérience utilisateur, nous utilisons une classe dérivée de AsyncTask évoquée précédemment. Ainsi, l'appel aux web services va être réalisé au sein d'un thread spécifique, ne bloquant pas l'UI Thread, au sein de la classe MeteoHomeLoader :

```
public class MeteoHomeLoader extends AsyncTask<Location, Void,
Meteo> {
    // ... champs, constructeur ...
```

```
   @Override
   protected void onPreExecute() {
      meteoActivity.showProgressBar();
   }

   @Override
   protected Meteo doInBackground(Location... params) {
      Meteo meteo = null;

      if (params.length > 0) {
         try {
            String city = CityGrabber.getCityFromLocation(
                    meteoActivity.getApplicationContext(),
params[0]);
            meteo = MeteoGrabber.getMeteoFor(city);
         } catch (Exception exc) {
            this.exc = exc;
         }
      }

      return meteo;
   }

   @Override
   protected void onPostExecute(Meteo result) {
      meteoActivity.hideProgressBar();

      if (exc == null) {
         meteoActivity.displayMeteo(result);
      }
   }

}
```

Cette tâche de chargement de la météo prend en entrée un objet Location, ne publie pas de résultats intermédiaires comme l'indique la classe Void et renvoie un objet Meteo en résultat final. Seul le contenu de la méthode surchargée *doInBackground* est exécuté dans un thread séparé. Ainsi, l'appel à l'activité MeteoActivity au sein de cette méthode est proscrit. On notera enfin que la publication de résultats intermédiaires peut se faire via *publishProgress* au sein de *doInBackground*. L'interaction avec le thread principal se faisant ensuite au sein de la méthode *onProgressUpdate*.

Localisation GPS

La position de l'utilisateur est rafraîchie grâce au service de localisation de l'OS accessible via la classe LocationManager. La première étape consiste à récupérer le meilleur fournisseur de position (le GPS dans le meilleur des cas) que ce service peut proposer grâce à la méthode getBestProvider. L'abonnement aux mises à jour de la position s'effectue ensuite en enregistrant un objet dérivant de LocationListener sur le LocationManager via requestLocationUpdates en précisant le fournisseur retenu. A chaque modification du positionnement, la méthode *onLocationChanged* est appelée sur l'objet de type *LocationListener* enregistré, avec en entrée la nouvelle position définie au sein d'un objet Location. Lorsque les mises à jour de positionnement ne sont plus nécessaires, il faut prendre garde à désabonner son listener via un *removeUpdates*, ce qui évitera de consommer inutilement l'énergie de la batterie du téléphone utilisateur.

Activité principale

L'ensemble des classes et méthodes nécessaires au bon fonctionnement de l'écran d'accueil de notre application développées, nous créons son activité principale MeteoActivity :

```
public class MeteoActivity extends Activity {
   // .. views ...

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.meteo);
      setupViews();
      installLocationManager();
   }

   @Override
   protected void onResume() {
      super.onResume();
      loadLocationFromLastKnown();
   }

   @Override
   protected void onPause() {
      super.onPause();
      removeLocationManager();
   }

   // … création du menu, récupération des views …

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
         case R.id.refreshPosition:
            locationManager.requestLocationUpdates(Util.get
BestProviderForLocation( locationManager, providerCriteria,
true);, 0, 0, fastMeteoLocationListener);
            return true;
         case R.id.listCities:
            Intent toListCitiesActivity = new Intent(this,
ListCitiesActivity.class);
            startActivity(toListCitiesActivity);
            return true;
         // ... autres entrées du menu ...
      }
   }

   private void installLocationManager() {
       locationManager = (LocationManager) getSystemService
(Context.LOCATION_SERVICE);
      providerCriteria = new Criteria();
      providerCriteria.setAccuracy(Criteria.ACCURACY_FINE);
   }
```

```
    private void loadLocationFromLastKnown() {
        Location location = null;
        String bestProvider = Util.getBestProviderForLocation
(locationManager,     providerCriteria, true);
        location = locationManager.getLastKnownLocation(best
Provider);
        loadMeteoForLocation(location);
    }

    private void removeLocationManager() {
       locationManager.removeUpdates(fastMeteoLocationListener);
    }

    private void loadMeteoForLocation(Location location) {
        new MeteoHomeLoader(this).execute(location);
    }

    public void displayMeteo(Meteo meteo) {
        // affichage de la météo …
    }
}
```

Le résultat de l'exécution de notre application est présenté au screenshot de la [Fig.4]. Au niveau de l'activité, on note la définition de l'IHM au sein d'un fichier XML menu.xml. Enfin, on remarque l'utilisation d'un Intent pour lancer l'activité *ListCitiesActivity* via la méthode *startActivity* de MeteoActivity. Pour des raisons de place, le contenu de cette seconde activité n'est pas détaillé ici mais techniquement parlant, elle n'est pas vraiment différente de l'activité principale MeteoActivity puisqu'elle se base sur les mêmes classes pour fonctionner.

Screenshot de l'application météo

Fichier AndroidManifest de l'application

Présent à la racine du projet, le fichier AndroidManifest.xml spécifie les caractéristiques de l'application. Il permet de préciser les permissions requises par l'application météo à savoir INTERNET pour les appels web services ainsi que ACCESS_FINE_LOCATION et ACCESS_COARSE_LOCATION pour le service de localisation. Cela permet à l'utilisateur de valider avant l'installation ce que l'application va pouvoir faire sur le téléphone. Dans le cas où l'application installée tenterait d'outrepasser ses droits, une erreur viendrait interrompre son exécution. D'autre part, on définit la version minimale du SDK et la version ciblée via la ligne :

```
<uses-sdk android:minSdkVersion="7" android:targetSdkVersion="8"/>
```

On cible donc la version 2.2 du SDK avec un support minimal pour la version 2.1. Enfin, les différentes activités y sont définies de même que l'activité principale. Seules ces activités seront considérées par l'OS lors de l'exécution de l'application, ce qui implique que le lancement d'une activité non référencée provoquera une erreur au runtime.

Tests

Comme pour les applications desktop ou web, les tests constituent un passage essentiel dans la réalisation d'une application Android. Ils peuvent être de différentes natures :

- Unitaires, testant l'exécution de méthodes au sein des classes de l'application
- Fonctionnels, testant le comportement général de l'application
- Matériels, qualifiant une application sur différents environnements

L'écosystème Android offre des solutions pour chacun de ces types de tests soit directement dans le SDK officiel soit via des bibliothèques tierces. Pour les tests unitaires automatisés, le plugin ADT facilite la création d'un projet de test pour un projet Android et utilise JUnit. En outre, le SDK amène un ensemble de classes dédiées aux tests permettant l'instrumentalisation du code Android donnant accès aux tests d'activités ou de services. Pour les activités, le cas de test réalisé doit hériter de la classe ActivityInstrumentationTestCase2<T> où T sera remplacé par le type d'activité testé. La mise en place de ces tests étant bien détaillée dans la littérature, nous ne nous y attarderons pas dans le cadre de cet article.

En revanche, nous mettons en place des tests fonctionnels automatisés grâce à l'API Robotium librement téléchargeable ici : http://code.google.com/p/robotium/. Sorte de Sélénium pour Android, Robotium propose une API simple et puissante qui complète les classes d'instrumentalisation du SDK pour combler certains manques. Le test fonctionnel mis en place va consister à lancer l'activité principale, vérifier que la météo d'une ville est bien affichée puis lancer l'activité de suivi des villes pour y ajouter la ville "Paris". Le test est conclu en vérifiant le chargement de la météo pour la ville ainsi saisie. Le code associé est présenté ci-dessous :

```
public class MeteoActivityFunctionalTest extends  Activity
InstrumentationTestCase2<MeteoActivity> {

private Solo solo;

    public MeteoActivityFunctionalTest() {
        super("com.app.fastmeteo", MeteoActivity.class);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
        // création objet Robotium
        solo = new Solo(getInstrumentation(), getActivity());
    }

    public void testFunctionalScenario() {
      Util.removeAllCityInCache(solo.getCurrentActivity().get
ApplicationContext());
        // attente du démarrage de l'activité
```

```
      solo.waitForActivity(this.getClass().getName(), 5000);
     TextView tv = (TextView) solo.getCurrentActivity().find
ViewById(
    com.app.fastmeteo.R.id.humidity);
       CharSequence cs = tv.getText();
       assertFalse("".equals(cs.toString()));

     solo.clickOnMenuItem(solo.getCurrentActivity().getString(
    com.app.fastmeteo.R.string.list_cities));
       solo.waitForActivity(ListCitiesActivity.class.getName
(), 5);
       // assert activité suivi des villes est chargée
       solo.assertCurrentActivity("ListCitiesActivity expected",
    ListCitiesActivity.class);

       // ajout d'une ville
     solo.clickOnMenuItem(solo.getCurrentActivity().getString(
    com.app.fastmeteo.R.string.add_city));
      ListCitiesActivity listCitiesActivity = (ListCitiesActivity)
solo.getCurrentActivity();
       AlertDialog dialog = (AlertDialog) listCitiesActivity
.getManagedDialogs().get(ListCitiesActivity.ADD_CITY_DIALOG);
       TextView commentText = (TextView) dialog.findViewById
(com.app.fastmeteo.R.id.commentText);
       commentText.requestFocus();
       solo.sendKey(KeyEvent.KEYCODE_P);
       solo.sendKey(KeyEvent.KEYCODE_A);
       solo.sendKey(KeyEvent.KEYCODE_R);
       solo.sendKey(KeyEvent.KEYCODE_I);
       solo.sendKey(KeyEvent.KEYCODE_S);

       solo.clickOnView(dialog.getButton(AlertDialog.BUTTON
_POSITIVE));
       // attente chargement météo
       solo.waitForActivity(ListCitiesActivity.class.getName
(), 5000);
     assertTrue(listCitiesActivity.getListAdapter().getCount
() == 1);
     Meteo meteo = (Meteo) listCitiesActivity.getListAdapter
().getItem(0);
      assertTrue("Paris".equalsIgnoreCase(meteo.getCity()));
       // nettoyage
     Util.removeAllCityInCache(solo.getCurrentActivity().get
ApplicationContext());
   }
}
```

Application FastMeteo publiée sur le market

La qualification de l'application sur différents environnements est réalisée grâce à l'émulateur Android inclus au sein du SDK. Un gestionnaire de machines virtuelles Android permet d'en créer autant que nécessaire en précisant pour chacune d'elles un grand nombre de caractéristiques allant de la version de l'OS à des caractéristiques hardware plus poussées telles que la taille et la résolution de l'écran, la présence d'un GPS ou d'un accéléromètre, … Le but par la suite étant de tester le comportement de l'application sur ces différents environnements au sein de l'émulateur. Bien entendu, ces tests ne sauraient remplacer entièrement une qualification sur différents terminaux physiques mais les moyens nécessaires dans ce cas là ne sont pas les mêmes !

Publication de l'application

La publication sur une place de marché est le point d'orgue de la réalisation d'une application Android. Plusieurs places de marché existent pour Android mais le market Android de Google reste et de loin la référence. La première étape consiste à créer un compte développeur sur le market en réglant les droits d'entrée à hauteur de 25 dollars ici : https://market.android.com/publish/. Ce dernier réglé, le développeur peut publier gratuitement autant d'applications que souhaité. Le packaging de l'application en APK est guidé par un assistant ADT au sein d'Eclipse. Durant l'export, il faut signer l'application en utilisant un certificat à réutiliser à chaque nouvel export qui sera réalisé par la suite. L'utilitaire de génération optimise le code du projet pour l'alléger et le rendre optimal avant sa publication. Le développeur va ensuite devoir importer son application packagée sur son compte développeur sur le market et préparer la publication en détaillant un certain nombre de points tels que son nom, une description de ce qu'elle fait et éventuellement son prix si elle n'est pas gratuite. En outre, il est obligatoire de fournir des copies d'écran donnant aux utilisateurs potentiels l'envie de la télécharger. Pour l'application météo nommée FastMeteo, la distribution se fait gratuitement et on peut constater la réussite de sa publication via l'espace personnel développeur du market [**Fig.5**]. C'est au sein de cet espace que les statistiques et les rapports d'erreur la concernant seront accessibles par la suite.

Conclusion

A travers le développement d'une application météo, cet article aura mis en exergue les différentes étapes du développement d'une application Android et permis de présenter les bases nécessaires à la réalisation d'applications Android. Un point essentiel étant de garder à l'esprit que les applications mobiles sont destinées à des terminaux mobiles, ce qui implique un certain nombre de contraintes pour garantir une performance optimale tout en minimisant la consommation d'énergie. Plus encore que pour des applications desktop ou web, la phase de tests est essentielle puisqu'elle va permettre de s'assurer du bon fonctionnement de l'application au niveau fonctionnel tout d'abord mais ensuite sur un ensemble de terminaux suffisamment représentatif des utilisateurs potentiels. Afin de tester le résultat obtenu, l'application FastMeteo développée au cours de cet article est disponible à l'adresse suivante : https://market.android.com/details?id=com.app.fastmeteo .

Sylvain **Saurel** – *Ingénieur d'Etudes Java / JEE*
sylvain.saurel@gmail.com

Réaliser des interfaces clientes riches avec les animations interpolées

Envie de donner un peu de vie à vos applications Android ? Grâce aux animations interpolées, cela devient possible.

Pour créer une tweened animation (Animation interpolée) sur une cible donnée (composant,vue…), nous devons utiliser la classe Animation capable d'interpréter les animation suivantes :

- RotateAnimation, permettant de faire tourner la cible ;
- ScaleAnimation, permettant de zoomer / dézoomer sur la cible
- AlphaAnimation, jouant sur la notion de transparence de la cible
- TranslateAnimation, permettant de déplacer la cible

Une animation peut être codée ou décrite dans un fichier XML. C'est ce second aspect que nous allons dans un premier temps aborder.

Secouez-moi !

Pour illustrer la notion d'animation interpolée, nous allons créer une animation secouant une zone de texte si celle-ci n'est pas renseignée. Pour commencer, créons un projet Android ayant pour nom « AppRichInterface ». Dans ce projet créons un nouveau fichier XML nommé « error » de type animation (File>new> Android XML File> Sélectionner le type « Animation »). La création de ce fichier a pour résultat la création d'un répertoire « anim » situé dans le dossier « res » contenant le fichier xml nouvellement créé. Nous pouvons à présent créer notre animation comme suit (Les commentaires ne sont qu'à titre informatif et ne doivent pas êtres inclus dans votre code sous peine de dysfonctionnement) :

```
<?xml version=»1.0» encoding=»utf-8»?>
<set xmlns:android=»http://schemas.android.com/apk/res/android»
android:shareInterpolator=»true»>


    <rotate
     android:fromDegrees=»0» // Degrés de rotation de départ
     android:toDegrees=»10» // Degrés d'inclinaison souhaités
     android:duration=»100» // Temps d'exécution en millisecondes
     android:pivotX=»50%» // Point pivot de la rotation sur l'axe des X
     android:pivotY=»50%» // Point pivot de la rotation sur l'axe des Y
     android:repeatCount=»3» // Nombre d'exécution de l'effet
    />


</set>
```

Comme nous pouvons le constater, le mouvement souhaité se fera à partir d'une animation de rotation dont le point pivot de la rotation est spécifié à 50% de l'axe des X et à 50% de l'axe des Y. Cela signifie que la rotation de l'objet se fera à partir de son centre. Répété trois fois, cela donne l'impression que l'objet cible de cette animation, est « secoué ».

Voyons à présent comment utiliser cette animation dans une activité. Pour cela prenons le code de l'activité de notre application et ajoutons les lignes suivantes :

```
package com.apprichinterface;



import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.EditText;



public class AppRichInterfaceActivity extends Activity {

   Animation errorAnimation;
   EditText textError;

 @Override
 public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);



     //Chargement de l'animation
     errorAnimation  =  AnimationUtils.loadAnimation(this,R.
anim.error);



     //Ajout d'un listener sur le bouton
     Button btnTest= (Button)findViewById(R.id.btnValidate);
     btnTest.setOnClickListener(new View.OnClickListener() {
       public void onClick(View v) {

          //Utilisation de la zone de texte
         textError = (EditText)findViewById(R.id.txtData);

         //Verification du contenu de la zone de texte
         //Si elle est vide, on joue l'animation
           if (textError.getText().toString() != «»)
```

```
            {
              playErrorAnimation(errorAnimation, textError);
            }
        }
      }
      );


      }


      private void playErrorAnimation(Animation runAnimation,
EditText target){

        target.startAnimation(runAnimation);
      }

}
```

Pour charger une animation, nous utilisons la méthode loadAnimation, recevant en paramètre l'identifiant du fichier XML décrivant l'animation à exécuter. Une fois l'animation chargée, nous l'exécutons sur le composant de vue via la méthode startAnimation.
Bien entendu, cela ne fonctionne pas si l'interface graphique n'est pas construite. Modifions donc le code du layout main.xml avec ces quelques lignes et exécutons l'application :

```
<?xml version=»1.0» encoding=»utf-8»?>
<LinearLayout xmlns:android=»http://schemas.android.com/apk
/res/android»
     android:orientation=»vertical»
     android:layout_width=»fill_parent»
     android:layout_height=»fill_parent»
     >
    <EditText android:id=»@+id/txtData» android:layout_height
=»wrap_content» android:layout_width=»match_parent»>
          <requestFocus></requestFocus>
     </EditText>
    <Button android:text=»Valider» android:id=»@+id/btnValidate»
android:layout_width=»wrap_content»   android:layout_height=
»wrap_content»></Button>
</LinearLayout>
```

Nous venons de voir comment créer une animation à partir d'un fichier XML. Voyons à présent comment écrire cette animation à l'aide du code java [Fig.1].

Secouez-moi en Java !

Reprenons le code de l'activité et remplaçons le code de création de l'animation par celui-ci

```
        //Les Pivots sont précisés à l'aide de la variable
RELATIVE_TO_SELF
         errorAnimation = new RotateAnimation(0, 10,
                          Animation.RELATIVE_TO_SELF, 0.5f,
                         Animation.RELATIVE_TO_SELF, 0.5f);
         errorAnimation.setDuration(100);
         errorAnimation.setRepeatCount(3);
```

Fig.1

Si nous exécutons de nouveau l'application, nous constatons que le comportement est identique.

Création de séquences d'animations

Les séquences d'animations peuvent être créées via le fichier XML en ajoutant successivement chaque animation devant être jouée ou par l'utilisation de la classe AnimationSet, comme le montre le code suivant :

```
package com.apprichinterface;



import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.AlphaAnimation;
import android.view.animation.Animation;
import android.view.animation.AnimationSet;
import android.view.animation.AnimationUtils;
import android.view.animation.RotateAnimation;
import android.widget.Button;
import android.widget.EditText;



public class AppRichInterfaceActivity extends Activity {


   AnimationSet errorSequenceAnimation;
   EditText textError;



 @Override
 public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);


     //    —— Creation de la séquence
```

```
errorSequenceAnimation = new AnimationSet(false);

RotateAnimation shakeAnimation = new RotateAnimation(0, 10,
        Animation.RELATIVE_TO_SELF, 0.5f,
        Animation.RELATIVE_TO_SELF, 0.5f);
shakeAnimation.setDuration(100);
shakeAnimation.setRepeatCount(3);
errorSequenceAnimation.addAnimation(shakeAnimation);

AlphaAnimation alphaAnimation = new AlphaAnimation(1, 0);
alphaAnimation.setDuration(1000);
//On précise quand doit être joué l'effet : à 300 ms après le
//début de la séquence
alphaAnimation.setStartOffset(300);
errorSequenceAnimation.addAnimation(alphaAnimation);



//Ajout d'un listener sur le bouton
Button btnTest= (Button)findViewById(R.id.btnValidate);
btnTest.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {

     //Utilisation de la zone de texte
    textError = (EditText)findViewById(R.id.txtData);

    //Verification du contenu de la zone de texte
    //Si elle est vide, on joue l'animation
    if (textError.getText().toString() != "")
    {


        //Utilisation de la séquence
            playErrorSequence(errorSequenceAnimation, textError);
          }
        }
      }
      );


   }


   private void playErrorSequence(AnimationSet runSequence,
EditText target)
    {
     target.startAnimation(runSequence);
    }


}
```

Par défaut, la classe AnimationSet exécute les animations en parallèle. Pour réaliser une séquence d'animation, il faut utiliser la méthode setStartOffset pour chaque animation afin de définir le timer de début d'exécution. Dans notre cas, l'animation alphaAnimation s'exécute 1000 millisecondes après le début de l'animation errorAnimation.

Voici donc un aperçu de ce que nous pouvons réaliser à l'aide d'animations interpolées, ce qui vous permettra de donner vie à vos applications.

■ Aurélien **Vannieuwenhuyze**
Architecte Flex / Mobilité
http://aurelien-vannieuwenhuyze.com

Robotium, l'outil de tests unitaires pour Android

Créé en janvier 2010, Robotium est un framework de tests unitaires dédié à la plateforme mobile Android. Dans cet article, nous allons aborder la prise en main de cet outil et vous allez vous rendre compte qu'il va vous devenir indispensable dans le processus de développement pour assurer la livraison d'applications Android qualitatives.

L'exemple décrit dans cet article a été réalisé à partir des éléments techniques suivants : SDK Android : Android_R12. Version de Robotium : Robotium-solo-2.5 (téléchargeable à cette adresse : http://code.google.com/p/robotium/)
Ces pré-requis sont importants car le cas pratique a été testé avec d'autres paramètres, notamment Android 2.2 et Robotium-solo-2.4, le comportement attendu n'était pas au rendez-vous.

Création d'un projet Android

Afin de comprendre le mécanisme de Robotium, il nous faut avant tout disposer d'un projet Android à tester.
Nous allons donc créer un projet que nous nommerons AppRobotium avec les paramètres suivants :

Paramètre	Valeur
Project Name	AppRobotium
Build Target	Android 2.3.3
Application name	AppRobotium
Package name	com.approbotium
Activity	AppRobotiumActivity
Min SDK 10 :	Correspondant à Android 2.3.3

L'objectif de l'application que nous allons développer est assez simple car il consiste à tester si un nombre saisi par l'utilisateur est pair ou impair [Fig.1].
Ceci se traduit par l'écriture du code dans les fichiers suivants.

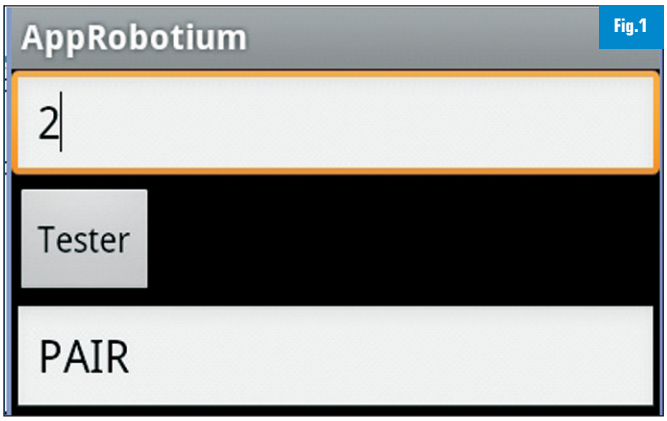


Fig.1

Le layout main.xml

```
<?xml version=»1.0» encoding=»utf-8»?>
<LinearLayout xmlns:android=»http://schemas.android.com/apk/res/android»
    android:orientation=»vertical»
    android:layout_width=»fill_parent»
    android:layout_height=»fill_parent»
    >
    <EditText android:layout_height=»wrap_content» android:layout_width=»match_parent» android:id=»@+id/txtNumber»>
        <requestFocus></requestFocus>
    </EditText>
    <Button android:text=»Tester» android:id=»@+id/btnTest» android:layout_width=»wrap_content» android:layout_height=»wrap_content»></Button>
    <EditText android:layout_height=»wrap_content» android:layout_width=»match_parent» android:id=»@+id/txtResult»></EditText>

</LinearLayout>
```

Classe AppRobotiumActivity :

```
package com.approbotium;


import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;



public class AppRobotiumActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);


        Button viewBtnTest = (Button)findViewById(R.id.btnTest);


        //Ajout d'un évènement sur le bouton de test
        viewBtnTest.setOnClickListener(new View.OnClick
Listener() {
           public void onClick(View v) {

              //Pair ou impair ?
              EditText txtNumber = (EditText)findViewById(R.
id.txtNumber);
               EditText txtResult = (EditText)findViewById(R.
id.txtResult);

                  if (txtNumber.getText().toString() != "")
                  {
                   Integer number = Integer.parseInt(txtNumber
.getText().toString());

                  if (number%2 == 1)
                  {
                    txtResult.setText("IMPAIR");
                  }
                  else
                  {
                    txtResult.setText("PAIR");
                  }
                  }
         }
        });



    }
}
```

Une fois la retranscription du code réalisée, nous pouvons vérifier le bon fonctionnement de l'application.

Utilisation de Robotium pour notre projet

Création du projet

Pour utiliser robotium, nous devons créer un nouveau projet de type AndroidTest (File>New > Projects > Android > Android Test Project) avec les paramètres suivants :

Paramètre	Valeur
Project Name	AppRobotiumTest
Test Target	Sélectionner le projet AppRobotium précédemment créé
Build Target	Android 2.3.3
Application name	AppRobotiumTest
Package name	com.approbotium.test
Min SDK 10 :	Correspondant à Android 2.3.3

Liaison avec le projet AppRobotium

La liaison entre le projet à tester (AppRobotium) et notre projet AppRobotiumTest s'effectue par la spécification du paramètre TestTarget du wizard de création de projet. Cette action se traduit par l'ajout d'une ligne dans le manifest de l'application AppRobotiumTest spécifiant le package à tester :

```
<instrumentation android:targetPackage=»com.approbotium»
android:name=»android.test.InstrumentationTestRunner» />
```

Dans notre cas, nous spécifions que l'application à tester est AppRobotium car elle se situe dans le package com.approbotium.

Ecriture des tests

Avant de procéder à l'écriture d'un test, il convient d'importer les librairies de Robotium 2.5 à notre projet :
>Clique droit sur le nom du projet > Properties > Java BuildPath > Add External Jar puis choisir robotium-solo-2.5.jar préalablement téléchargée.

La classe de tests

L'écriture des tests et du scénario de test se réalise par le biais d'une classe (que nous nommerons ClassTest) dont voici le code :

```
package com.approbotium.test;

import com.approbotium.AppRobotiumActivity;

//Import de la bibliothèque Robotium
import com.jayway.android.robotium.solo.Solo;
import android.test.ActivityInstrumentationTestCase2;


//Les tests vont être réalisés sur l'activité AppRobotium
Activity
public class ClassTest extends ActivityInstrumentationTest
Case2<AppRobotiumActivity> {

 private Solo solo;

 //Constructeur spécifiant le package et l'activité à tester
 public ClassTest()
 {
   super("com.approbotium", AppRobotiumActivity.class);
 }
```

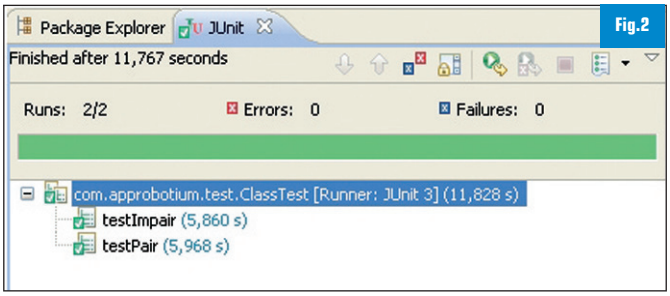


Fig.2

```
//Methode de préparation des tests (Executée avant chaque test)
//On peut notamment y spécifier une connexion à une base de
donnée etc...
@Override
protected void setUp() throws Exception {
  super.setUp();
  this.solo = new Solo(getInstrumentation(), getActivity());
}


//Methode s'assurant que l'environnement est propre avant de
passer
//au test suivant : variables nettoyées, objets détruits...
//(Executée avant chaque test)
@Override
protected void tearDown() throws Exception{
  try {
    this.solo.finalize();
    } catch (Throwable e) {
    e.printStackTrace();
    }
    getActivity().finish();
    super.tearDown();
}



//————————— TEST ————————————/


public void testPair() {

  //Ajout d'une donnée dans le premier champ texte (Index 0)
  solo.enterText(0, «2»);

  //Clique sur le bouton tester
  solo.clickOnButton(«Tester»);

  //Mise en place du test
  //Vérifie que nous avons bien le mot PAIR affiché à l'écran
  assertTrue(solo.searchText(«PAIR»));

}


public void testImpair() {
  solo.enterText(0, «3»);
  solo.clickOnButton(«Tester»);
  assertTrue(solo.searchText(«IMPAIR»));

 }



}
```

Dans cette classe, nous avons spécifié deux tests :
• TestPair (Vérification qu'un nombre saisi est pair)
• TestImpair (Vérification que le nombre saisi est impair)
En analysant le code de ces tests, nous constatons que Robotium est assez simple à appréhender. En effet via la méthode enterText(), nous somme en mesure de faire saisir du texte par Robotium dans une zone de texte précise. La méthode clickButton permet de simuler un click sur un bouton présent sur l'interface. Enfin, l'instruction assertTrue nous permet de tester si un cas de test est vrai : `assertTrue(solo.searchText(«PAIR»))` se traduisant par « Il existe la chaîne de caractère PAIR dans l'interface graphique ».

Exécution des tests.
L'exécution des tests s'effectue par un clic droit sur le projet AppRobotiumTest puis par le choix du menu contextuel « run as » et en terminant par le choix de l'option « Android JUnitTest ». Se présente alors à nous une interface de suivi d'exécution des tests et leurs résultat [Fig.2]. A noter que pour exécuter à nouveau un test (en cas d'erreur par exemple), il suffit de réaliser un clic droit sur celui-ci et choisir l'option « run ».

Pour aller plus loin

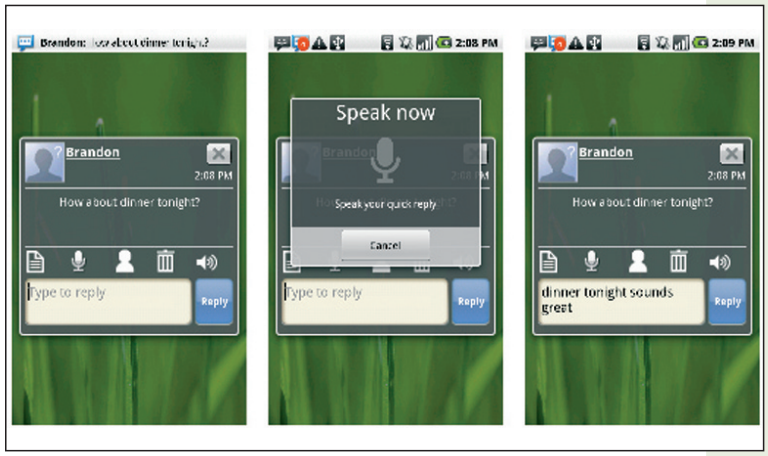
Cet exemple est assez minimaliste et ne permet pas d'appréhender toutes les fonctions de Robotium. En effet, avec un seul article dédié à la présentation de Robotium, il ne nous est pas possible de décrire toutes les possibilités de cette librairie. Afin de remédier à cela, j'ai écris un petit projet, téléchargeable sur le site de Programmez, permettant de tester l'appel de menu contextuels, de vérifier la suppression d'éléments dans une liste, de naviguer dans les activités... ce qui vous permettra de mettre en place Robotium sur vos projets Android.

Aurélien **Vannieuwenhuyze**
Architecte Flex / Mobilité
http://aurelien-vannieuwenhuyze.com

Faites parler vos applications Android

Dans cet article, nous allons découvrir comment utiliser TextToSpeech dans une application Android, afin de lui donner une dimension d'interaction supplémentaire avec l'utilisateur.

La synthèse vocale se réalise grâce à l'API TTS (Text To Speech). Le code ci-dessous démontre comment l'utiliser par un exemple basique permettant de faire parler l'application qui sera chargée de nous dire « Bonjour Programmez ». Bien entendu, il convient avant tout de créer un projet Android (AppSpeaking) et dans l'activité principale d'y insérer le code suivant :



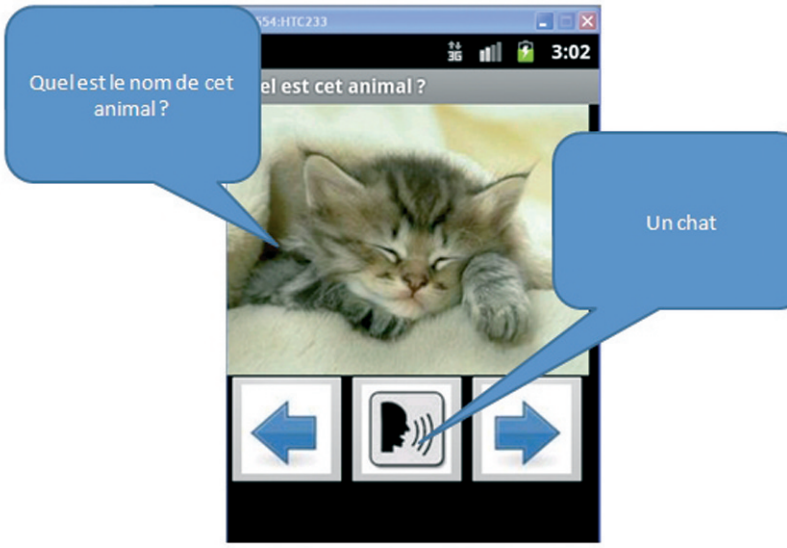

```
package com.appspeaking;

import java.util.Locale;

import android.app.Activity;
import android.content.Intent;
import android.speech.tts.TextToSpeech;
import android.speech.tts.TextToSpeech.Engine;
import android.speech.tts.TextToSpeech.OnInitListener;
import android.os.Bundle;

public class AppSpeakingActivity extends Activity {

//Variable
 private static int TTS_DATA_CHECK = 1;
 private TextToSpeech tts = null;
 private boolean ttsIsInit = false;


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

//Initialisation de la librairie Text To Speech
initTextToSpeech();

    }


    //Methode d'initialisation
    private void initTextToSpeech()
    {
   //Appel du moteur Text To Speech, avec pour action
première la vérification de la présence des bibliothèques nécessaires
        Intent intent = new     Intent(TextToSpeech.Engine.
ACTION_CHECK_TTS_DATA);
        startActivityForResult(intent, TTS_DATA_CHECK);
    }
```

```
    //Résultat de l'initialisation
    protected void onActivityResult(int requestCode, int result
Code, Intent data){

     if (requestCode == TTS_DATA_CHECK) {

       //Si les bibliothèques TTS (Text to speech) existent
       if (resultCode == Engine.CHECK_VOICE_DATA_PASS)
       {
           //On initialise le text to speech
           tts = new TextToSpeech(this, new OnInitListener() {

         @Override
         public void onInit(int status) {
           if (status == TextToSpeech.SUCCESS)
           {
             ttsIsInit = true;
             if (tts.isLanguageAvailable(Locale.FRANCE)>=0)
               tts.setLanguage(Locale.FRANCE);

             tts.setPitch(0.8f);
             tts.setSpeechRate(1.1f);
             speak();

           }


         }
       });
       }

     }
     //Si les bibliothèques TTS n'existent pas on les installe
     else
```

```
      {
        Intent installVoice = new Intent(Engine.ACTION_INSTALL
_TTS_DATA);
        startActivity(installVoice);
      }
     }



     //Méthode permettant de faire parler l'application
     private void speak()
     {
      if (tts != null && ttsIsInit)
      {
        tts.speak(«Bonjour Programmez !», TextToSpeech.QUEUE
_ADD, null);
      }
     }


     @Override
     public void onDestroy() {

      if (tts !=null)
      {
        tts.stop();
        tts.shutdown();
      }

      super.onDestroy();
     }

}
```

A la lecture de ces lignes nous pouvons définir le processus d'utilisation de la librairie text to speech comme suit :

- On initialise l'appel au moteur TextToSpeech
- On vérifie que les bibliothèques existent via l'action ACTION_CHECK_TTS_DATA de la classe Textspeech.Engine qui renvoie 1 si ces dernières sont présentes
- Dans le cas où elles sont installées, onActivityResult recevra CHECK_VOICE_DATA_PASS = 1 ce qui confirmera la présence des bibliothèques et ce qui nous permettra de paramétrer l'API. Si les librairies n'existent pas, nous demanderons leur installation via l'action Engine.ACTION_INSTALL_TTS_DATA



La librairie étant présente, nous pouvons à dès lors configurer l'API via les méthodes

- setLanguage servant à spécifier la langue à utiliser ;
- setPitch précisant le ton de la voix (0,1 à 2);
- setSpeechRate spécifiant la vitesse de lecture du texte (;

Nous pouvons à présent tester notre application sur un téléphone Android ou bien sur l'émulateur à condition d'avoir installé les librairies TTS sur ce dernier. A titre personnel, je préconise l'utilisation d'un device matériel afin de s'affranchir de tout problème d'installation et de configuration.

Maintenant que nous savons comment utiliser l'API text to speech, nous pouvons l'utiliser dans diverses applications.

D.R.

Un exemple d'application

A titre d'exemple, je vous propose une petite application destinée à nos chérubins. Une image d'animal s'affiche à l'écran et pose la question suivante « Quel est le nom de cet animal ?». En cliquant sur le bouton de réponse, l'application indique à l'utilisateur l'image en cours de visualisation.

Afin de ne pas surcharger cet article visant à présenter l'utilisation de l'API text to speech, vous trouverez l'ensemble des sources de cette application sur le site du magazine.

Libre à vous d'en modifier le contenu mais également de créer votre propre application avec vos nouvelles connaissances.

■ Aurélien **Vannieuwenhuyze**
Architecte Flex / Mobilité
http://aurelien-vannieuwenhuyze.com

Comment monétiser son application Android ?

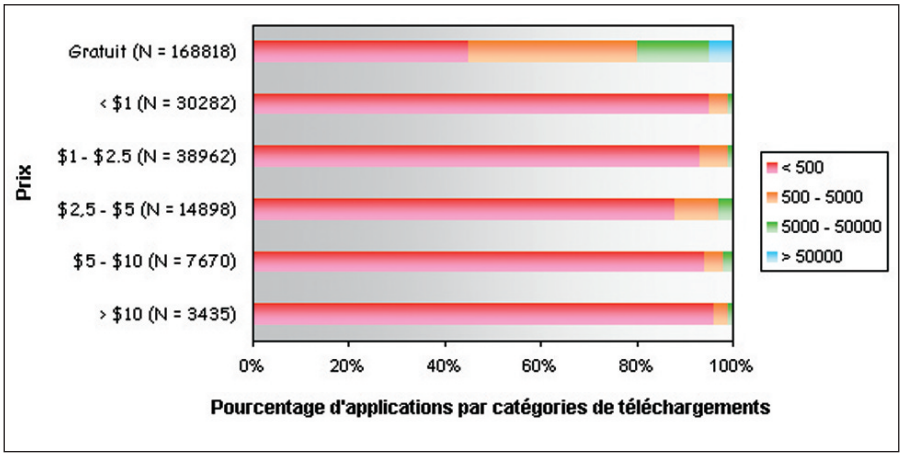
Le développement d'applications mobiles en général, et Android en particulier, attire de plus en plus de développeurs fascinés par la réussite de quelques applications stars dont les créateurs sont aujourd'hui devenus millionnaires. Si la réussite extraordinaire de ces applications reste un épiphénomène, plutôt rare, de type success story, il existe bel et bien un certain nombre de possibilités pour monétiser une application Android. Cet article se propose de faire un tour d'horizon des principaux business models existant à l'heure actuelle dans le domaine.

Devenu l'élément dominant du monde des smartphones, l'écosystème Android voit arriver un nombre croissant de développeurs désireux de développer des applications mobiles. Pour faciliter l'accès de la plateforme aux développeurs Java, Google met à disposition tout une gamme d'outils allant d'un émulateur de terminal au SDK complet, en passant par des outils de développement basés sur Eclipse, sans oublier une documentation officielle de grande qualité. De fait, le développeur arrivant sur la plateforme Android a vite fait d'imaginer des idées d'applications à développer en vue d'une publication au sein d'un market Android. Dès la phase de conception, le développeur est confronté à une question essentielle : Comment monétiser son application ? Ceci afin de rentabiliser le travail effectué pour le développement de celle-ci. A ce jour, il existe plusieurs places de marché pour les applications Android en sus du market officiel proposé par Google. La différence principale entre celles-ci se situant dans la part reversée au développeur par le fournisseur du market. Dans ce qui suit, nous nous intéressons essentiellement au market Android de Google, qui est la référence en la matière, bien qu'une distribution sur plusieurs places de marché soit une idée envisageable. Après 3 ans d'existence, les retours d'expérience concernant ce market fleurissent et font ressortir 4 business models majeurs pour la monétisation d'une application Android.

Modèle de l'application payante

Le premier d'entre eux est le plus simple à mettre en place et consiste à fixer un prix à son application lors de sa publication sur le market. Afin de pouvoir la télécharger, l'utilisateur final doit payer ce prix. Ce modèle peut paraître le plus rentable puisqu'il permet d'avoir une rentrée d'argent directe et fixe pour chaque téléchargement. Néanmoins, il est bon de garder à l'esprit que le monde Android souffre en quelque sorte d'une image de gratuité véhiculée par le côté ouvert de la plateforme au niveau technique. Ainsi, si les téléchargements d'applications sont de plus en plus nombreux, la répartition des téléchargements par catégories de prix (figure 1) montre clairement que les applications gratuites sont très largement majoritaires (près de 65%), ce qui a tendance à accentuer l'habitude de gratuité des utilisateurs finaux d'applications Android.

On constate que les plus gros succès (plus de 50 000 téléchargements) concernent uniquement des applications gratuites ! En outre, Les prix des applications payantes les plus téléchargées ne dépassent pas les 5 dollars, ce qui implique de vendre un gros volume d'applications pour arriver à quelque chose de très rentable. Et sur ces quelques deniers récoltés, le développeur ne perçoit au final que 70% puisque le reste revient à Google dans le cadre du market officiel Android. Il faut donc bien se rendre à l'évidence, l'utilisateur Android n'aime pas payer pour une application avant de l'avoir pleinement testée et s'il le fait, ses critiques risquent d'être acerbes en cas de déception. Au final, le modèle du paiement à l'achat semble être intéressant uniquement pour des applications qui bénéficient d'un effet buzz très important et qui de fait arriveront à percer les classements du market malgré leur prix.



Modèle de l'application gratuite limitée

On l'a dit, l'utilisateur Android n'aime pas payer directement une application sans avoir pu la tester de manière approfondie. Certes, le market propose à l'utilisateur de

pouvoir annuler l'achat d'une application si elle ne le convainc pas, mais cela reste plus que limité dans le temps. De fait, de nombreux développeurs ou sociétés vont proposer une version gratuite mais allégée de leur application afin de la faire découvrir à l'utilisateur et de lui donner envie d'acheter la version payante dont le contenu est pleinement fonctionnel. La difficulté avec ce modèle réside dans la différenciation entre l'expérience gratuite de l'application et la version payante. Il faut donc proposer suffisamment de fonctionnalités à l'utilisateur pour qu'il accroche à l'application tout en laissant assez d'options manquantes pour qu'il investisse dans la version payante venant les combler. Ce modèle est celui qui a été adopté par le jeu Angry Birds qui constitue une réelle success story en la matière pour l'univers Android. Dans ce dernier cas, la société a en outre su tirer parti du succès initial du jeu pour augmenter les revenus via la sortie de versions spécifiques du jeu lors de chaque grande fête. Un bon coup marketing.

Modèle financé par la publicité

Le financement d'une application par la publicité constitue également un modèle très utilisé, essentiellement pour des applications gratuites. Il consiste à intégrer de la publicité dans son application afin de générer des revenus qui dépendent de la régie publicitaire choisie. Il existe 3 types de rémunération :

- Le Coût par Action (CPA) qui paie uniquement quand le clic sur la publicité conduit ensuite à une action (un achat le plus souvent). C'est le plus rémunérateur.
- Le Coût par Clic (CPC) est un peu moins rémunérateur mais est le plus utilisé. La régie paie lorsqu'un clic est effectué sur la publicité affichée.
- Le Coût par Mille (CPM) propose un prix par millier d'affichages de la publicité. C'est clairement le moins rémunérateur.

Le taux de clic demeure pour le moment légèrement supérieur à ce qui se fait sur le web mais des outils de blocage de ces publicités se développent, ce qui peut venir annihiler les espoirs de monétisation placés dans ce modèle. La publicité n'est pas toujours bien vue par l'utilisateur, mais il l'acceptera d'autant mieux qu'elle est utilisée avec parcimonie sans détérioration du confort d'utilisation et sans risques de clics involontaires. Ce modèle demeure intéressant pour monétiser une application sans faire payer l'utilisateur et est facile à mettre en place avec le framework Admob proposé par Google par exemple. Néanmoins, il ne permet pas de présager d'un revenu fixe pour son application puisque celui-ci sera éventuellement perçu au cours du temps si les téléchargements de l'application sont nombreux, ce qui est favorisé par son côté gratuit.

Modèle avec contenu payant

Le dernier modèle est sûrement celui qui reste le moins exploité jusqu'à présent mais semble promis à un bel avenir. Il s'appuie sur une application gratuite au sein de laquelle du contenu va pouvoir être acheté. Celui-ci peut être de différentes formes : dans le cas de jeux, l'utilisateur pourra par exemple acheter des niveaux, des solutions ou des biens virtuels permettant de progresser au sein du jeu. En outre, cela peut également concerner l'achat de contenu sur des sites d'informations. Pour ce modèle, Google propose là encore un outil de paiement utilisable directement dans son application et lié au market Android. La grande difficulté et le grand risque de ce type de modèle vont être de réussir à créer une application qui poussera l'utilisateur à acheter du contenu. Dans le cas où ce serait un échec, l'application développée serait mise gratuitement à disposition des utilisateurs sans aucune rentabilisation du travail effectué. En revanche, dès lors que l'utilisateur voit un intérêt à acheter du contenu au sein de votre application, ce modèle peut devenir très rentable et présente l'avantage non négligeable de générer des revenus récurrents contrairement au paiement à l'achat. La possibilité prochaine d'intégrer ces paiements au sein d'applications directement dans la facture opérateur de l'utilisateur devrait permettre de mieux intégrer ces achats et d'éviter les éventuelles annulations de dernière seconde provoquées par un nombre de validations trop important pour un achat.

Conclusion

Les quelques succès extraordinaires qu'a connu la plateforme Android ont tendance à faire croire aux développeurs qu'ils tiennent, grâce au développement d'applications mobiles, un nouvel eldorado qui va leur assurer de gros revenus si ce n'est la fortune. Si les différents business models pour monétiser une application Android commencent à être éprouvés et les possibilités de rentabilisation réelles, il n'en demeure pas moins que très peu de développeurs et de sociétés arrivent à tirer leur épingle du jeu au final.

Les recettes du succès ne tiennent pas tant dans le choix du bon business model que dans la qualité de l'application produite et dans les besoins auxquels elle vient répondre. Pour les développeurs pensant tenir une application de cet acabit, cet article constitue un point d'entrée intéressant quant au choix d'un modèle de monétisation tout en gardant à l'esprit qu'il demeure possible de mixer les différentes approches présentées.

■ Sylvain **Saurel**
Ingénieur d'Etudes Java / JEE
sylvain.saurel@gmail.com

Les outils des Décideurs Informatiques

Vous avez besoin d'info sur des sujets d'administration, de sécurité, de progiciel, de projets ? Accédez directement à l'information ciblée.

www.solutions-logiciels.com

☐ **OUI**, *je m'abonne* (écrire en lettres capitales)

Envoyer par la poste à : Solutions Logiciels, service Diffusion, GLIE - 17 chemin des Boulangers 78926 Yvelines cedex 9 - ou par fax : 01 55 56 70 20

1 an : 50€ *au lieu de 60€, prix au numéro (Tarif France métropolitaine) - Autres destinations : CEE et Suisse : 60€ - Algérie, Maroc, Tunisie : 65€ , Canada : 80€ - Dom : 75€ Tom : 100€*

10 numéros par an.

☐ M. ☐ Mme ☐ Mlle Société .....

Titre : ..... Fonction : ☐ Directeur informatique ☐ Responsable informatique ☐ Chef de projet ☐ Admin ☐ Autre .....

NOM ..... Prénom .....

N° ..... rue .....

Complément .....

Code postal : |_|_|_|_|_| Ville .....

Adresse mail .....

☐ *Je joins mon réglement par chèque à l'ordre de SOLUTIONS LOGICIELS* ☐ *Je souhaite régler à réception de facture*

Editeur : K-Now sarl au cap de 8000€ siège social : 21 rue de Fécamp, 75012 Paris

Interview d'Eric de Saqui de Sannes, DRH de SOGETI FRANCE

« Nous prévoyons 2000 recrutements en 2012, dont 60% de jeunes diplômés »

SOGETI est une société internationale de services informatiques, présente dans 15 pays avec plus de 100 implantations en Europe, aux Etats-Unis et en Inde. Filiale de CapGemini, Sogeti compte environ 20 000 salariés, avec un recrutement majoritaire en France ou l'effectif atteint 10 000 collaborateurs.

Quel est votre volume de recrutement prévu pour 2012 ?

Nous prévoyons un peu plus de 1 500 profils à l'échelon national sur l'année 2012, plus de 2 000 en intégrant SOGETI High-Tech. Une dynamique favorable laisse présager un dépassement de ces objectifs. En termes de répartition, nous pouvons compter 60% de jeunes diplômés pour 40% de profils confirmés ou experts. En complément des méthodes classiques de recrutement, SOGETI FRANCE High Tech organise régulièrement, dans toutes les régions, des rencontres avec les candidats. Dans ce cadre, nous avons récemment organisé une soirée « Speed-Recruiting » à Issy-les-Moulineaux, relayée par les réseaux sociaux. Cela a été un franc succès, confirmé par une assez bonne corrélation de profils avec les postes recherchés : Ingénieurs nouvelles technologies, Administration système ou consultants SI, par exemple.

Quels sont les profils que vous recherchez et les types de postes à créer ?

Nous recherchons, de façon récurrente, des profils expérimentés et confirmés dans les nouvelles technologies - Java/J2EE et .Net. Des compétences en sécurité informatique, idéalement avec une bonne expérience, sont recherchées régulièrement, compte tenu de notre positionnement.

SOGETI est également leader dans le domaine d'activité des tests logiciels, et prédit un développement de cette activité dans les prochaines années. Nous recherchons donc des ingénieurs et consultants en maintenance système et testing, qualification et certification. Des profils compétents en études, Maîtrise d'Ouvrage et Conseil, en intégration et développement et en mise en production et sécurité sont activement recherchés. Enfin, en décisionnel, nous recrutons des profils destinés aux métiers Application Services, experts et consultants.

Portez-vous une part importante de votre recrutement sur les jeunes diplômés ?

SOGETI recrute des jeunes diplômés en se tournant notamment vers les universités et les écoles d'ingénieurs ou de commerce. Plus de 70% des collaborateurs de SOGETI sont issus de ces écoles.

Proposez-vous des formations ?

SOGETI possède son propre institut de formation. Il faut savoir que 6% de la masse salariale est destinée chaque année à la formation. De plus, un réseau social d'entreprise, véritable plateforme collaborative, permet d'aller chercher de l'information ou de résoudre certains problèmes via un outil de support collaboratif partagé par les salariés. C'est également un outil idéal pour l'intégration des nouveaux entrants.

Qu'est-ce que vous pourriez envoyer comme message à un candidat qui voudrait postuler chez vous ?

Toujours engagé dans l'innovation, SOGETI s'efforce de rester une entreprise à taille humaine, et propose à ses salariés un véritable suivi de carrière, plusieurs modules d'E-Learning, des possibilités de télétravail et même une conciergerie d'entreprise. En 2009, une ouverture à 4% du capital a été proposée aux collaborateurs.

■ Propos recueillis par Thierry **Lellouche**

Ingénieurs développement :

Il faut compter 10 000 ingénieurs au total, en intégrant SOGETI High-Tech. Sur les métiers traditionnels du développement, plus de 1 000 ingénieurs font partie de nos effectifs.

Chefs de projet

Environ 600 collaborateurs

INDICATEUR MENSUEL APEC
DES OFFRES D'EMPLOI CADRE

Dans le secteur de l'informatique, 11 561 offres d'emploi ont été publiées sur le site de l'APEC en Août, ce qui représentait 28% des offres totales.

Sur l'ensemble des secteurs, de septembre 2010 à août 2011, 494 556 offres d'emploi ont été publiées, ce qui représente une progression de 67%. L'informatique représente sur cette période 137 916 offres, en croissance de 64% par rapport à 2010. Le tableau présente les demandes par type de fonctions.

	Août 2011		en cumul sur 12 mois glissants		
Source APEC	Nb offres	2011/10	Nb offres	2011/10	Structures (en %)
Direction informatique	327		3908	+ 108%	
Exploitation, maintenance informatique	643		7249	+ 49%	
Informatique de gestion	3877		50817	+ 44%	
Informatique industrielle	1510		16700	+ 54%	
Informatique web, sites et portails internet	1545		17498	+ 123%	
Maîtrise d'ouvrage et fonctionnel	1210		14965	+ 87%	
Système, réseaux, données	2449		26779	+ 74%	
INFORMATIQUE	**11561**	**+ 48%**	**137916**	**+ 64%**	**28**

Eclipse rame, comment y remédier ?

Mon métier m'amène à côtoyer plusieurs équipes d'informaticiens développant en Java. Dans quasiment 100% des cas, l'IDE de prédilection est Eclipse. Visant à satisfaire les besoins de tout un chacun, les DSI, souvent via une cellule d'architecture transverse, proposent très fréquemment une installation maison d'Eclipse incluant divers plugins. L'effort est louable, mais m'a malheureusement paru trop souvent contre-productif.

Allant à contre-courant, je vais donc vous faire part de mon utilisation d'Eclipse au sein des diverses équipes où j'ai œuvré.

Avoir la dernière version

Le problème de base d'un packaging maison d'Eclipse est habituellement son obsolescence. Le packaging fut coûteux à produire et l'équilibre des plugins fragile. On ne peut donc pas le refaire tous les jours. La version vieillit et les plugins aussi. Eclipse sortant une version mineure par saison, cela se produit même plus rapidement que l'on pense. De mon côté, je commence par prendre la dernière version JEE disponible. Je prends la version JEE, mais ne l'utilise en fait que très peu. Principalement pour les fonctionnalités XML. J'ai toujours trouvé WTP particulièrement instable. Je mets à jour mon package tous les 6 mois environ. Souvent, avec un simple Check for update. N'allez toutefois pas penser que je vis sur la corde raide. Même si Indigo vient de sortir, je vais quand même attendre un peu avant de l'installer. Le temps d'avoir la mise à jour des plugins. Je prends aussi bien soin de travailler moi-même quelque temps avec une nouvelle version avant de la donner à mes développeurs.

Mettre très peu de plugins

En deuxième place des erreurs les plus fréquemment rencontrées est l'amoncellement de plugins installés. On tente de faire plaisir à tout le monde, mais très vite, le package se retrouve doté d'une myriade de plugins en tout genre. Le pauvre Eclipse ne sait plus où donner de la tête. Il monte déraisonnablement en usage mémoire et rame. Trop de plugins veulent faire trop de choses à chaque clic et se gênent entre eux. Il est lent au démarrage, disparaît spontanément plusieurs fois par jour et le menu contextuel contient tellement d'options qu'on ne sait plus quoi en faire. Laissez tomber ! Inutile de satisfaire tout le monde. Ne mettez que quelques plugins très utiles et laissez les équipes en ajouter un ou deux de plus si besoin. Mieux, proposez-leur une liste dans laquelle ils feront leur marché.

Pour vous donner une idée, voici ma liste classique de plugins :

- **m2eclipse** : Cette intégration de Maven dans Eclipse a longtemps été instable et lente. Elle est maintenant un projet Eclipse à part entière et fonctionne sans soucis même sur des projets de bonne taille
- **MoreUnit** : Facilite la rédaction des tests. Les raccourcis ajoutés sont particulièrement utiles (saut entre le test et la classe testée, création automatique de la classe de test si absente et dernièrement, création des mocks)
- **eGit** ou **Subclipse** : Intégration de Git ou SVN en fonction du gestionnaire de sources utilisé. Pour Subclipse, le connecteur JavaHL m'a toujours paru plus stable et cohérent que SVNKit.
- …et c'est tout

Une dernière chose sur les plugins. Des outils, tels que Eclipse Memory Analyzer, sont très utiles et existent en version plugin et RCP. Utilisez la version RCP. Inutile de surcharger et déstabiliser votre Eclipse de développement pour un usage ponctuel de MAT.

Apprendre les raccourcis clavier

S'il y a une chose qui mérite que vous perdiez votre temps à approfondir sous Eclipse, c'est bien les raccourcis clavier. C'est le pas nécessaire vers une meilleure productivité.

Prenez le temps de parcourir périodiquement la liste disponible dans les préférences [Fig.1]. Quelques raccourcis très utiles :

Raccourci	Description
F3	Ouvrir la classe du type sélectionné
F4	Ouvrir la vue de hiérarchie de classe du type sélectionné
Ctrl+Shift+G	Affiche les appelants de l'objet sélectionné
Ctrl+R (MoreUnit)	Lance le test lié à la classe ouverte ou le test (si la classe ouverte est un test)
Ctrl+T	Ouvre la liste des implémentations de l'interface sélectionnée. Pratique pour naviguer d'une implémentation concrète à l'autre dans une architecture de dépendances par interface
Ctrl+Shift+T	Trouver une classe. Pour ne pas avoir à fouiller péniblement dans l'arborescence des packages. À noter que la recherche est très bien faite et supporte les astérisques et ne spécifier que la première lettre de chaque mot en majuscule (EMS va vous trouver EasyMockSupport) Il trouve les classes aussi bien dans le projet que dans ses dépendances
Ctrl+Shift+R	Trouver une ressource. En fait n'importe quel fichier du workspace. Ne cherchez pas dans les jars en dépendance toutefois
Ctrl+K	Trouver la prochaine occurrence dès la dernière recherche ou du mot sélectionné
Ctrl+D	Effacer la ligne courante
Ctrl+L	Aller à un numéro de ligne
Ctrl+1	Quickfix. Très très utile. Essayez-le sur chaque erreur ou avertissement dans le code. Il vous fera gagner énormément de temps. Vous allez même constater qu'il est même parfois plus efficace de créer sciemment une erreur et d'utiliser le quickfix que de taper correctement
Ctrl+E	Pour naviguer entre les fichiers ouverts
Alt+Shift+T	Fait apparaître les refactoring disponibles pour la sélection
Alt+Shift+S	Fait apparaître le menu contextuel de génération de source
Alt + Up/Down	Déplacer la ligne courante ou les lignes sélectionnées vers le haut ou le bas dans le fichier
Ctrl+Shift+L	Affiche la liste de tous les raccourcis disponibles selon la position du curseur

Utiliser les vues

Avec le temps, Eclipse s'est enrichi de plusieurs vues très utiles et méconnues. Je n'ai pas la prétention de les connaître toutes, mais en voici quelques-unes que j'apprécie particulièrement.

Expressions

Cette vue est un complément à la vue Variables. Elle permet l'évaluation instantanée d'une liste d'expressions plus ou moins complexes. Un exemple serait myList.get(i) dans une boucle permettant de connaître l'objet courant à chaque itération [Fig.2].

Breakpoints

Tout le monde connaît la vue Breakpoints mais assez peu de gens connaissent les Exception Breakpoints. Il s'agit d'un point d'arrêt se déclenchant au lancement d'une exception dans le code. Indispensable pour trouver l'endroit où « ça plante » [Fig.3].

Je vous suggère de parcourir les divers boutons de cette vue. Ils sont tous utiles. Par exemple, le Suspend thread / Suspend VM indique si le point d'arrêt doit suspendre uniquement le thread où il survient ou bien l'entière JVM. Par défaut, seul le thread du point d'arrêt, est suspendu, mais cela complexifie le débogage, car l'application continue à recevoir des appels en arrière-plan. La solution est donc d'arrêter toute la JVM.

Display

Une autre vue particulièrement puissante. Elle permet l'évaluation ou l'affichage du résultat de n'importe quel bout de code. Par exemple, si, sur un point d'arrêt, vous faites ex.printStackTrace(), vous verrez la stack trace apparaître sur la console [Fig.4]. Il est possible d'exécuter n'importe quel bout de code. Si les classes nécessaires ne sont pas importées par la classe où vous avez mis votre point d'arrêt, vous pouvez la référencer en mettant le nom complet incluant le package.

Remote System Explorer

Il s'agit en fait d'une perspective entière et non d'une vue. Elle permet de vous connecter à un système distant (par exemple une machine Unix) et :

- Parcourir l'arborescence de fichiers
- Ouvrir un fichier, le modifier et le sauvegarder
- Lancer un terminal sur la machine à distance et passer des commandes.

Une véritable session X dans votre Eclipse [Fig.5].

Autres

Il en existe bien d'autres pouvant répondre à un besoin ponctuel. Allez vite jeter un coup d'œil à la liste (Window -> Show View -> Other) [Fig.6].

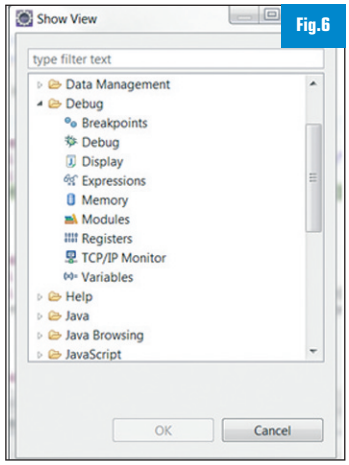


Fig.6

Bien configurer ses paramètres de workspace et de projet

Eclipse possède énormément d'options de configuration. En général, si vous vous dites « *je déteste quand Eclipse fait ça* », il y a sûrement une option pour

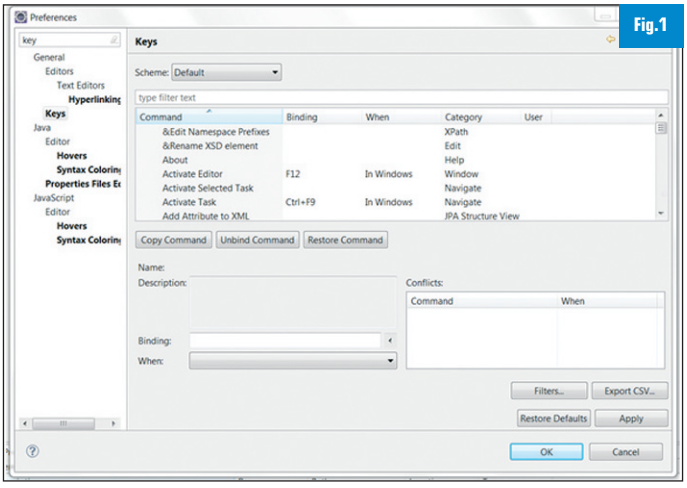


Fig.1

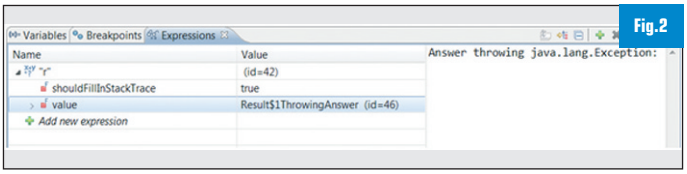


Fig.2

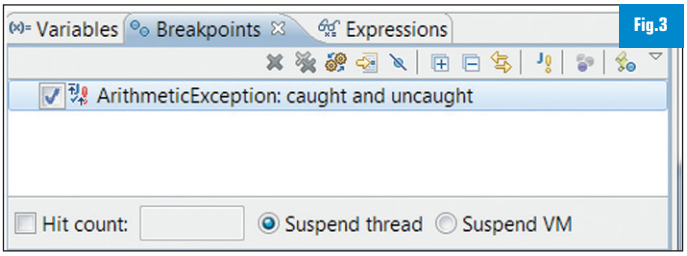


Fig.3

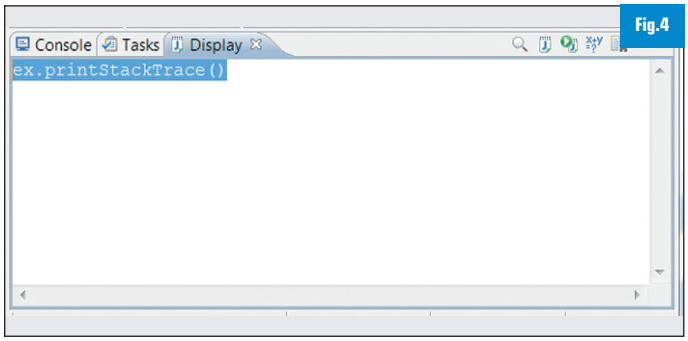


Fig.4

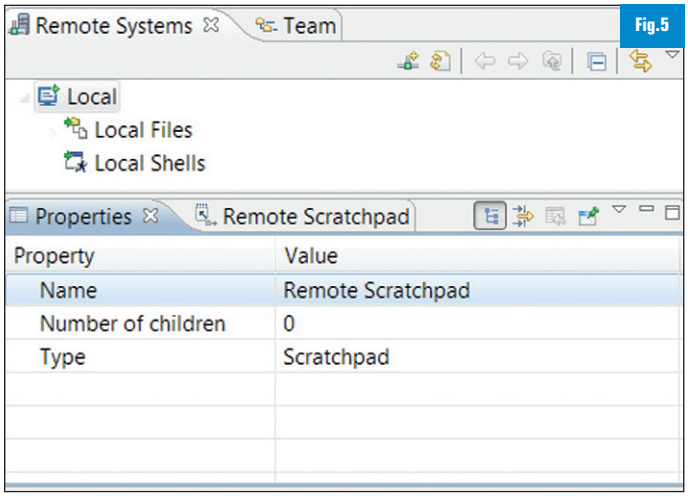


Fig.5

lui dire d'arrêter. Il est aussi très important de comprendre le système de préférences à deux niveaux. Vous avez les préférences du workspace et celles par projet.

Celles par projets sont gardées dans des fichiers situés dans le répertoire .settings du projet. Elles peuvent donc être mises dans votre gestionnaire de source. Ensuite, encore une fois, mes options préférées. Elles sont données pour Java, mais sont souvent aussi disponibles pour d'autres langages.

- **Java ->Editor->Content Assist->Favorites** : Permet à l'autocomplétion d'Eclipse de trouver les méthodes de vos imports statiques favoris. Vous y mettez par exemple org.junit.Assert et org.easymock.EasyMock.
- **Java->Editor->Save Actions** : Spécifie des opérations à effectuer systématiquement à la sauvegarde d'un fichier. Y mettre le formatage du code et l'optimisation des imports est fortement recommandé. Si vous n'êtes pas content du résultat suite à une sauvegarde, un Ctrl+Z l'annulera.
- **Java->Code Style->Code Templates** : Ces templates sont utilisés par le quick fix pour générer le code. Il est très utile de les modifier pour ne pas avoir à supprimer systématiquement les TODO et avoir donc une meilleure correction.
- **Java->Code Style->Organize Imports** : Pour trier vos imports. J'y mets Number of static imports needed for .* à zéro pour m'assurer des imports statiques en .*
- **Java->Editor->Content Assist** : Vous en avez assez qu'Eclipse insère devant l'ancienne méthode le nom de la nouvelle choisie dans la combo d'autocomplétion au lieu de remplacer l'ancienne? Changer Completion inserts pour Completion overwrites. Je vous suggère aussi *Insert best guessed arguments* qui devine nettement mieux les paramètres que vous voudriez passer à une méthode que *Insert parameter names*.
- **Java->Editor->Hover** : Eclipse fait apparaître de nombreuses infobulles lorsque vous promenez votre souris. Je les trouve particulièrement agaçantes et cet écran d'options vous permet de désactiver ceux que vous n'aimez pas.

Ces options ne sont qu'un court résumé des possibilités. Ma recommandation si vous cherchez à configurer quelque chose est d'aller dans les préférences et d'utiliser le champ de recherche (« type filter text »). Il est très efficace. Et n'oubliez surtout pas le paramétrage le plus important : La configuration de votre formateur de code (Java->Code Style->Formatter).

Ne pas perdre votre temps à optimiser les paramètres JVM

C'est un sport beaucoup trop populaire. Mieux vaut dépenser son temps à apprendre des raccourcis et à travailler sur la configuration du workspace. Je n'ai jamais constaté d'impacts notoires de performance en changeant les options JVM du eclipse.ini d'une version récente d'Eclipse. La seule exception est d'augmenter la mémoire maximale permise suite à l'installation de plusieurs plugins. Mais comme mentionné au début de l'article, il faut probablement commencer par retirer des plugins…

Conclusion

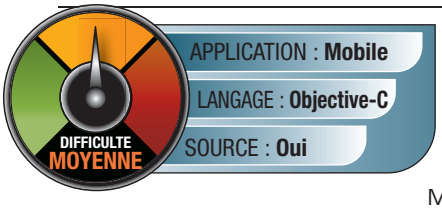
Je ne pense pas détenir l'ultime configuration ni être omnipotent. J'essaie d'être pragmatique. Il est rare sur un projet d'avoir assez de temps libre pour pouvoir essayer tous les plugins et toutes les configurations d'Eclipse. Il faut donc arbitrer et viser le meilleur retour sur investissement. J'ai tenté de vous donner quelques clés. J'espère qu'elles vont seront utiles dans votre recherche de productivité. Et surtout n'oubliez pas, allez à l'essentiel et voyagez léger. Eclipse ne s'en portera que mieux.

■ Henri **Tremblay**
Architecte senior chez OCTO Technology, Il est aussi le principal développeur d'EasyMock et Objenesis. Il a 11 ans d'expérience en informatique, principalement en Java, mais aussi en C/C++/C#.

Devenir un développeur iOS

2e partie

Dans notre précédent article, nous avons vu quelles démarches sont nécessaires auprès d'Apple pour pouvoir développer et déployer une application sur iPhone et iPad, à savoir souscrire un abonnement à l'Apple Developer Program.



Nous avons également abordé le principe le plus important du développement d'applications, à savoir le design pattern MVC (Modèle-Vue-Contrôleur), qui nous a permis de créer notre première application de type « Hello World ». Le temps est donc venu de créer une application qui ne se contentera pas simplement de vous dire bonjour mais qui contiendra plusieurs écrans, permettra de naviguer entre ces écrans, affichera des messages d'information et bien plus encore.

QUEL TYPE DE CONTRÔLEUR POUR QUEL USAGE ?

Comme nous l'avons évoqué précédemment, le développement d'applications iOS repose sur MVC qui nécessite d'implémenter un ou plusieurs contrôleurs pour pouvoir piloter son application, chaque contrôleur étant généralement associé à une vue (un écran).

La première question que va donc devoir se poser le développeur va concerner le contrôleur à utiliser pour l'écran d'accueil de son application, sachant qu'il en existe plusieurs types au sein d'iOS. Les 4 principaux quand nous parlons de créer une application permettant d'afficher des écrans et éventuellement de naviguer entre ceux-ci sont :

- UITableViewController
- UINavigationController
- UISplitViewController
- UITabBarController

Il est bon de noter que de nouveaux contrôleurs sont ajoutés à iOS au fil des versions, iOS 5 ajoutera notamment le « **UIPageViewController** » permettant de créer des applications ayant la même ergonomie que l'application « **iBooks** ».

Bien que le choix puisse être changé à n'importe quel moment, celui-ci peut être fait dès la création de l'application, lors du choix du type de projet [Fig.1].

Comme nous pouvons le voir sur la capture d'écran [Fig.1], différents projets dédiés à chaque contrôleur peuvent être choisis :

- Navigation-based application
- Split View-based Application
- Tab Bar Application

Pour vous donner quelques exemples d'applications utilisant ces différents modèles, l'application « **Mail** » sur iPhone correspond au modèle « **Navigation based** » qui permet de pouvoir naviguer d'écran en écran et revenir en arrière via la barre de navigation présente en haut de l'écran avec le bouton « Retour ».

La même application sur iPad correspond pour sa part au modèle « **Split View-based** ». Lorsque vous utilisez l'iPad en mode portrait (vertical), l'application ressemble beaucoup à celle de l'iPhone. En revanche en mode paysage (horizontal), l'écran est séparé en 2 sections, celle de gauche listant les messages et celle de droite affichant le contenu des messages.

Pour le 3e modèle, il correspond à l'application « **Horloge** » de l'iPhone qui permet de naviguer entre différentes fonctionnalités (Horloges, Alarme, Chronomètre et Minuteur) via la barre de navigation présente en bas de l'écran et qui affiche une icône et un titre pour chaque fonctionnalité.

Il faut toutefois noter que nombre d'applications utilisent des combinaisons de différents contrôleurs pour implémenter l'ergonomie souhaitée. Si nous reprenons l'exemple de l'application « **Mail** », elle utilise à la fois un « **UINavigationController** » pour gérer la navigation entre les écrans, et un « **UITableViewController** » pour afficher la liste des messages. Nous reviendrons ultérieurement sur cette possibilité. Pour la suite de cet article, nous allons démarrer notre application à partir du projet « Navigation-based Application ». Créez une application comme expliqué lors de l'article précédent. Pour notre exemple, nous allons créer une application permettant de gérer une liste de contacts.

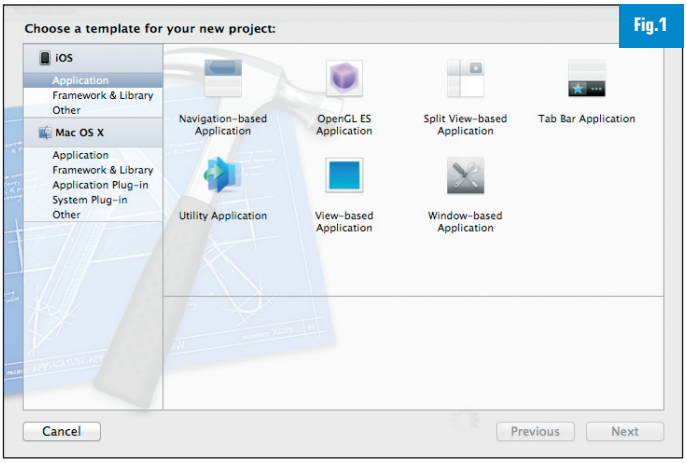


Fig.1

ANATOMIE DE NOTRE APPLICATION

Notre application reprend plus ou moins la même structure que celle créée lors de notre précédent article. Vous devez notamment avoir des fichiers nommés xxxAppDelegate.h et xxxAppDelegate.m (xxx correspond au nom que vous avez donné à votre application) qui représentent le point d'entrée de votre application.

Vous devez également avoir les fichiers suivants :

- MainWindow.xib
- RootViewController.xib
- RootViewController.h
- RootViewController.m

Commençons par ouvrir le fichier « **MainWindow.xib** » qui correspond au fichier de description de l'interface graphique principale de votre application. Vous devriez voir apparaître l'interface permettant de créer de manière visuelle votre interface.

Si vous regardez avec attention la partie gauche de l'interface, différentes icônes représentant notamment des cubes en vue filaire, vue pleine, vue semi-transparente, sont présentes. La dernière de ces

icônes contenant une flèche grise sur fond jaune correspond au contrôleur principal de votre application (**Navigation Controller**). Si vous cliquez sur cette icône, le contrôleur apparaîtra dans l'interface pour pouvoir être modifié [Fig.2].

Comme nous pouvons le voir, le contrôleur comprend une barre de navigation grise, ainsi qu'une vue qui sera chargée à partir du fichier nommé « **RootViewController** ». Si vous sélectionnez la vue et que vous demandez d'afficher l'inspecteur d'attributs (aller dans le menu *View > Utilities > Show Attributes Inspector*), vous constaterez que vous pouvez sélectionner quel contrôleur sera chargé automatiquement dans cette vue lors de l'exécution de votre application (*NIB Name = RootViewController*).

Le contrôleur principal de notre application allant essayer de charger un second contrôleur au sein de la vue de l'écran d'accueil, regardons à quoi ressemble celui-ci en ouvrant le fichier nommé **RootViewController.xib** à partir de la liste des fichiers théoriquement présente sur la partie gauche de l'environnement Xcode.

Lorsque vous ouvrez ce fichier, vous devriez voir apparaître le contrôle classique des applications iOS qui permet de lister des éléments en permettant de les faire défiler vers le haut ou le bas (ex : les musiques de l'application Musique). Ce contrôle se nomme « UITableView ».

Maintenant que nous avons fait rapidement le tour de notre application et alors que nous n'avons encore rien modifié dans le code, celle-ci est toutefois déjà fonctionnelle. Si vous l'exécutez (allez dans le menu *Product > Run*), vous devriez arriver au résultat de la [Fig.3].

Comme vous pouvez le constater, bien que notre application s'exécute sans problème, celle-ci ne sert pas à grand-chose puisque nous ne pouvons pas naviguer dedans et que la liste d'information est vide. Voyons donc comment pallier tout ceci.

LE CONTRÔLE « UITABLEVIEW » ET SON CONTRÔLEUR

Avant de voir comment utiliser le contrôle UITableView et le contrôleur qui lui est associé, faisons un peu de cosmétique dans notre application en ajoutant un titre à notre barre de navigation principale. Pour cela, retournez sur le fichier MainWindow.xib, sélectionnez le contrôleur de navigation et cliquez sur la barre de navigation grise présente en haut de la fenêtre représentant l'écran de votre application. Une fois la barre de navigation sélectionnée, affichez l'inspecteur d'attributs (menu *View > Utilities > Show Attributes Inspector*) pour voir apparaître les propriétés de celle-ci. Dans l'inspecteur, vous devriez voir apparaître 3 zones éditables nommées « *Title* », « *Prompt* » et « *Back Button* ». Pour donner un titre à votre barre de navigation, il vous suffit simplement de remplir la zone nommée « Title ». La fenêtre de design de l'écran de votre application est mise à jour en temps réel pour refléter les modifications que vous avez effectuées.

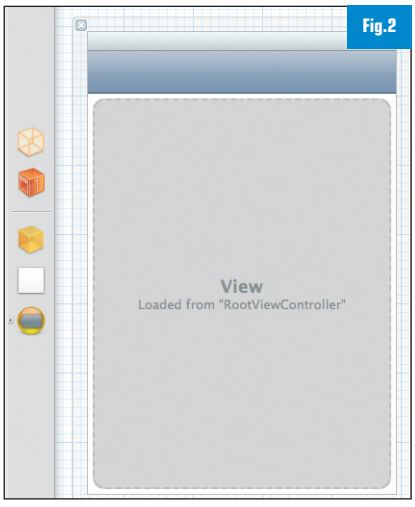


Fig.2

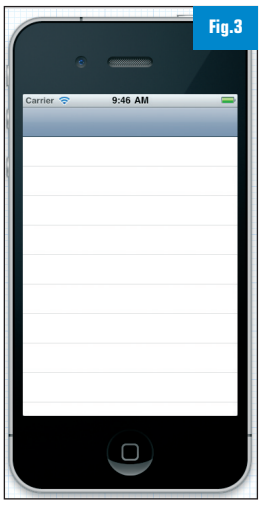


Fig.3

Passons maintenant à notre contrôle de liste et voyons comment faire en sorte que celui-ci affiche des données. Pour cela, retournez dans le fichier « *RootViewController.xib* » et sélectionnez, dans la partie gauche du designer, le cube ayant des arêtes jaunes. Pour être sûr que c'est celui qui nous intéresse, laissez le curseur de la souris quelques secondes au-dessus. Vous devriez voir apparaître une info-bulle indiquant « *File's Owner* ».

Une fois la sélection effectuée, faites apparaître l'inspecteur d'identité (menu *View > Utilities > Show Identity Inspector*). Dans celui-ci, vous devriez voir indiqué que le type de la classe associée à ce contrôleur est du type « *RootViewController* ». Cela signifie que lors de l'exécution de votre application, une instance de cette classe sera créée et que le comportement de votre écran sera délégué à cette instance. Si vous ouvrez le fichier nommé « RootViewController.h » qui correspond à la définition de votre contrôleur, vous devriez avoir quelque chose ressemblant à ce qui se trouve ci-dessous.

```
#import <UIKit/UIKit.h>
@interface RootViewController : UITableViewController
@end
```

Cette définition permet de déclarer une nouvelle classe nommée « *RootViewController* », qui surcharge la classe de base « *UITableViewController* ». Comme vous pouvez le constater pour le moment, bien que nous ayons surchargé la classe, aucun attribut ou propriété supplémentaire n'a été ajoutée à la classe. Nous verrons ultérieurement si nous avons besoin d'ajouter ce type d'éléments.

Le fait que dans notre application, la classe « *RootViewController* » qui sera instanciée à l'exécution, hérite de la classe « *UITableViewController* », aura pour principal intérêt de pouvoir piloter le comportement du contrôle « *UITableView* » que nous avons vu précédemment. Retournons sur notre fichier « RootViewController.xib », en sélectionnant le contrôle « UITableView » demandons cette fois de faire apparaître l'inspecteur de connexions (menu *View > Utilities > Show Connections Inspector*).

Vous devriez voir apparaître 3 connexions identiques à celles-ci :

- View File's Owner
- DataSource File's Owner
- Delegate File's Owner

Ces 3 connexions sont extrêmement importantes car ce sont d'elles, dont va dépendre le bon fonctionnement du contrôleur et du fait que des données soient affichées ou non dans le contrôle « *UITableView* ». La première connexion permet d'indiquer au contrôle, qui sera en charge de gérer la vue associée au contrôle lors de l'exécution de celui-ci. Vu que la connexion indique que c'est le « File's Owner », cela signifie que c'est le contrôleur qui sera en charge de cette gestion. La seconde connexion permet au contrôle de savoir à qui s'adresser pour récupérer les données devant être affichées dans le contrôle. Une nouvelle fois, il s'agira du contrôleur.

La troisième et dernière connexion permet au contrôle de savoir à qui seront destinés les évènements qui seront déclenchés par celui-ci. Là encore, le contrôleur sera en charge d'effectuer cette ges-

tion. Ces connexions sont importantes car elles représentent le fondement du pattern MVC dont nous avons parlé dans notre précédent article. Toute la logique de fonctionnement d'un écran doit être en théorie exécutée par le contrôleur. Ces connexions sont donc là pour faire le lien entre la vue, notre contrôle « *UITableView* », et le contrôleur (le « *RootViewController* » qui hérite du « *UITableViewController* »).

IMPLÉMENTATION DU « ROOTVIEWCONTROLLER »

Maintenant que nous avons passé en revue les liens existants entre la vue et le contrôleur, voyons comment faire en sorte que celui-ci puisse afficher des données. Pour cela, commençons par ouvrir le fichier « *RootViewController.m* », qui correspond à l'implémentation du contrôleur, et voyons ce qu'il contient.

```
#import "RootViewController.h"

@implementation RootViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
}
- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
}
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
}
- (void)viewWillDisappear:(BOOL)animated
{
 [super viewWillDisappear:animated];
}
- (void)viewDidDisappear:(BOOL)animated
{
 [super viewDidDisappear:animated];
}
-  (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation
{
 return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
}
- (void)viewDidUnload
{
    [super viewDidUnload];
}
- (void)dealloc
{
    [super dealloc];
}
@end
```

Les lignes précédentes représentent un tronc commun que l'on retrouvera sur tous les contrôleurs de vue et qui permettent de traiter un ensemble d'évènements déclenchés par iOS, et auxquels le développeur peut répondre pour adapter le fonctionnement de son application en conséquence.

Un des évènements le plus fréquemment utilisé est celui nommé « viewDidLoad » et qui permet, une fois la vue instanciée et toutes les options par défaut initialisées, de pouvoir personnaliser le comportement de celle-ci. Nous pourrions par exemple changer des propriétés de la vue telles que la couleur d'arrière-plan ou bien ajouter des vues enfants. Nous aborderons ces exemples dans un prochain article. Comme vous pouvez le voir, de très nombreux évènements peuvent être traités et notamment ceux ayant trait à la gestion mémoire (didReceiveMemoryWarning / dealloc) ou bien à la gestion de l'orientation du périphérique (shouldAutorotateToInterfaceOrientation) pour déterminer si l'application peut fonctionner en mode portrait ou paysage lorsque l'utilisateur tourne son iPhone / iPad.

Regardons maintenant en détail les 3 évènements vitaux au fonctionnement de notre contrôle « UITableView ».

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
    return 0;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

   UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
       cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell.
    return cell;
}
```

Le premier évènement nommé « *numberOfSectionsInTableView* » permet de déterminer combien de groupes d'éléments seront créés au sein de notre liste pour organiser les données d'un point de vue logique. Le meilleur exemple de ce mode de fonctionnement est l'application de gestion des réglages des périphériques iOS qui est entièrement basée sur le contrôle « UITableView » et qui contient différentes sections.

Le deuxième évènement nommé « *numberOfRowsInSection* », permet d'indiquer pour chaque section, combien de lignes doivent être créées et seront donc présentées à l'utilisateur lorsque celui-ci naviguera au sein des données de la liste. Pour savoir quelle section va

être traitée par le contrôle, vous voyez qu'il existe un paramètre du type « *NSInteger* » et nommé « *section* » sur lequel vous pouvez vous appuyer pour implémenter une logique particulière (ex : section 1 = 10 lignes et section 2 = 5 lignes).
Le troisième et dernier évènement nommé « *cellForRowAtIndexPath* » est probablement le plus important, puisque c'est lui qui sera en charge de créer les cellules qui seront affichées au sein de votre liste. Le paramètre nommé « indexPath » permet de déterminer quelle section et quelle cellule au sein de la section doit être créée. Cela permet de pouvoir gérer plusieurs types de cellules au sein d'une même liste. Comme nous le voyons avec le code généré par Xcode lors de la création de notre projet, notre liste contiendra par défaut 1 section, chaque section contenant 0 ligne. Commençons donc par augmenter le nombre de lignes à 10. Le code qui a été généré par Xcode permet de réaliser tout ce qui est nécessaire pour créer et afficher les cellules. Il ne nous reste plus qu'à insérer à la place du commentaire « *Configure the cell* », le code additionnel que nous souhaitons. Prenons comme exemple d'afficher le numéro de la cellule.

```
- (UITableViewCell *)tableView:(UITableView *)tableView cell
ForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

   UITableViewCell *cell = [tableView dequeueReusableCellWith
Identifier:CellIdentifier];
    if (cell == nil) {
       cell = [[[UITableViewCell alloc] initWithStyle:UITable
ViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell.
   cell.textLabel.text = [NSString stringWithFormat:@"Cellule
#%d", indexPath.row];

    return cell;
}
```

Bien que désormais pleinement fonctionnelle, attardons-nous quelques instants sur une portion particulière de la méthode :

```
UITableViewCell *cell = [tableView dequeueReusableCellWith
Identifier:CellIdentifier];
if (cell == nil) {
      cell = [[[UITableViewCell alloc] initWithStyle:UITable
ViewCellStyleDefault reuseIdentifier:CellIdentifier] autorelease];
}
```

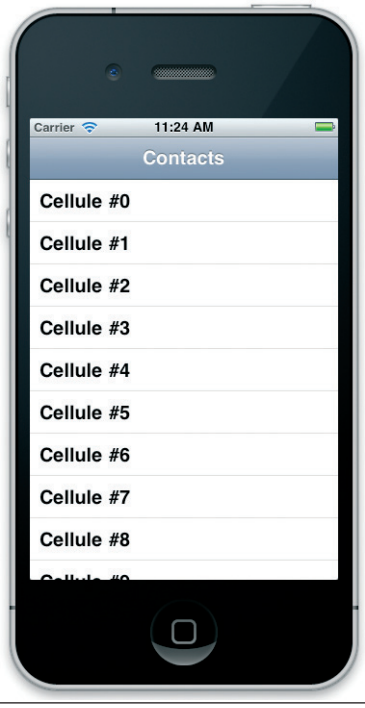
Comme vous pouvez le constater, nous ne créons pas directement une cellule à chaque fois que la méthode est appelée pour des questions de performance. En effet, dans un exemple simple comme le nôtre avec 10 lignes, il n'y aurait pas trop de soucis. En revanche sur des listes qui contiennent des centaines ou des milliers de cellules, les ressources nécessaires seraient trop importantes pour garder toutes les cellules créées en mémoire.
iOS gère donc un système de mise en cache ou seules les cellules visibles à l'écran subsistent en mémoire. L'appel à la méthode « *dequeueReusableCellWithIdentifier* » permet de vérifier sur une cellule ayant été créée au préalable, et dont la même n'a pas encore été libérée, qu'elle ne pourrait pas être réutilisée pour minimiser l'impact sur la mémoire nécessaire à notre application pour s'exécuter. Dans le cas ou différents types de cellules seraient utilisés au sein d'une même liste, le système de cache est géré pour chaque type. C'est à cela que sert le paramètre « *CellIdentifier* », qui permet d'attribuer un identifiant à un type, et donc de demander à iOS de vérifier dans le système de cache qui nous intéresse. Si nous exécutons maintenant notre application, nous arrivons au résultat [Fig.4] ou nous voyons que notre barre de navigation possède un titre, et que notre liste possède le nombre d'éléments que nous avons spécifié dans la méthode « *numberOfRowsInSection* ».
Un dernier mot sur les méthodes présentées ci-dessus. Comme vous pouvez le voir, chacune d'entre elles possède en paramètre une référence vers le contrôle « UITableView ». Vous pourriez vous dire que cela ne sert à rien puisque notre écran ne possède qu'une liste. C'est effectivement le cas ici mais nous pourrions imaginer des applications plus complexes ou un écran possèderait plusieurs listes, ces listes étant gérées par le même contrôleur. Dans ce cas, le fait de savoir quel évènement concerne quel contrôle est alors un élément crucial.

CONCLUSION

Comme nous l'avons vu dans cet article, de nombreux contrôleurs existent pour créer vos applications et mettre en œuvre l'ergonomie qui vous semble la plus adaptée à base de barres de navigation, de listes, de navigation par onglet, etc.
Nous avons également vu en détail comment implémenter le contrôleur « UITableViewController » afin de gérer une liste de données.
Dans un prochain article, nous verrons comment gérer les évènements des contrôles (listes, boutons…) et comment naviguer entre différents écrans au sein de notre application en tirant parti des possibilités des différents contrôleurs de navigation.

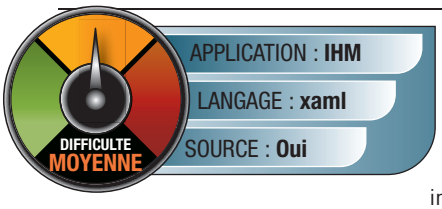
Stéphane **Cordonnier**
Directeur – iTouch - http://www.itouchconsulting.com

Fig.4



Les Behaviors et les TriggerActions

Les Behaviors et les TriggerActions sont des outils proposés par Expression Blend, permettant de réaliser du code interactif avec une approche intéressante. C'est aussi une des passerelles puissantes entre le travail du développeur et celui du designer en WPF et en Silverlight. Nous nous attacherons dans cet article à décrire ces classes et à proposer des exemples simples d'utilisation.



Il arrive souvent que le développeur interagisse directement avec l'interface graphique et par conséquent implémente du code interactif. L'interaction avec l'interface depuis le code peut entraver la collaboration car le Designer n'est plus maître pour faire évoluer l'IHM et le développeur fige d'une certaine manière le comportement interactif de l'application.

L'introduction du pattern M-V-VM (Model-View-ViewModel) permet de regagner en souplesse et évolutivité mais, malgré tout, les spécificités graphiques nécessitent l'intervention du développeur sur la couche visuelle. D'autre part, le pattern M-V-VM permet d'une certaine manière d'utiliser les tests unitaires sur le ViewModel pour tester la logique applicative. Or, l'implémentation de code touchant à la couche graphique dans le ViewModel limite l'usage des tests unitaires sur le ViewModel. Grâce aux mécanismes du Behavior et du TriggerAction, le code interactif peut être isolé afin d'améliorer sa réutilisabilité et éviter qu'il ne crée une adhérence entre les couches graphiques et la logique applicative, redonnant ainsi tout son sens à la collaboration Designer – Développeur.

GÉNÉRALITÉS TECHNIQUES

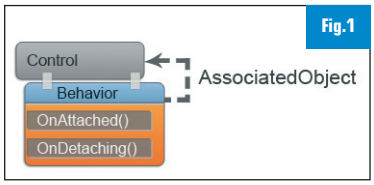
Techniquement parlant, les Behaviors et les TriggerActions sont des classes génériques issues des librairies livrées avec Expression Blend permettant d'attacher des bouts de codes à un Contrôle (à un DependencyObject de manière générale). L'association de ces instances avec des contrôles utilise la mécanique des AttachedProperties. Fonctionnellement parlant et grâce à l'excellente intégration dans Expression Blend, ces classes permettent de créer des objets interactifs, modulaires, réutilisables et paramétrables qui vont permettre d'ajouter des interactions dans la Vue à un type de contrôle (grâce à la généricité) sans passer par de l'héritage ou du « code behind ». En outre - nous le verrons au cours de cet article - il est possible de faire communiquer ces objets entre eux, c'est pourquoi, il est souvent approprié d'implémenter autant que possible des comportements élémentaires. Un ultime avantage des Behaviors et des TriggerActions est d'atomiser les comportements en objets élémentaires et donc de rendre maintenable le code interactif souvent conséquent et chaotique par des voies plus traditionnelles.

LES BEHAVIORS

Comme son nom l'indique, la Behavior est une classe qui permet de coder un comportement. Pour cela, la classe abstraite Behavior propose d'implémenter deux méthodes virtuelles que sont :

- **« OnAttached() »**: qui se produit lorsque la Behavior est attachée à l'objet associé (avant l'évènement onLoad dans le cas où cette Behavior est attachée statiquement par exemple)
- **« OnDetaching() »** : qui se produit lorsque la Behavior est en train de se détacher de l'objet associé.

La classe Behavior expose également une propriété **« AssociatedObject »** qui contient l'instance de l'objet sur lequel la Behavior a été attachée [Fig.1].

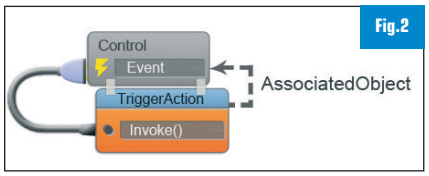


La Behavior prévient donc de son attachement et de son détachement et permet d'accéder à l'instance associée. Elle dispose de tout ce dont le développeur a besoin pour coder n'importe quel type de comportement.

Si le développeur dispose d'une grande liberté pour coder son comportement à sa guise et sans contrainte particulière, il convient toutefois de respecter au moins une règle importante pour éviter les fuites mémoire : il est nécessaire de déconstruire tout ce qui a été construit et qui pourrait persister dans le « OnDetaching » (exemple : les abonnements aux évènements).

LES TRIGGERACTIONS

Bien que cela ne soit pas vrai techniquement, nous pouvons considérer la TriggerAction comme un sous-ensemble de la Behavior: Là ou sur la Behavior le développeur avait la liberté de faire ce qu'il souhaite en code quand il le souhaite, la TriggerAction ne fait "que" exécuter une action sur un évènement choisi par l'utilisateur de la TriggerAction [Fig.2]. La Behavior associe un comportement à un type d'objet ; la TriggerAction associe un comportement déclenché sur un événement pour un type d'objet.



BEHAVIORS ET TRIGGERACTIONS LES PLUS COURANTS

Expression Blend offre nativement un large choix de Behaviors et TriggerActions (notez au passage que par convention dans Blend, une TriggerAction est toujours suffixée par « Action » ce qui permet de les reconnaître facilement).

La liste ci-dessous reprend les plus courants avec une petite explication de leurs fonctions :

- CallMethodAction : Appelle une méthode d'un objet.
- ChangePropertyAction : Permet de changer ou incrémenter une propriété d'un objet tout en définissant une transition si nécessaire.
- ControlStoryboardAction : Permet de contrôler un StoryBoard
- GoToStateAction : Affiche un état visuel.
- InvokeCommandAction : Invoque une commande (ICommand).

MouseDragElementBehavior : Permet de déplacer un élément visuel avec la souris.

UTILISATION DU GOTOSTATEACTION

Le rôle du GotoStateAction consiste à afficher un état visuel (VisualState). Cette classe étant une TriggerAction, ce changement s'effectue logiquement au déclenchement d'un évènement de l'objet associé. Soit une Vue « MainView » et une autre « PopupView » avec pour chacun d'eux un ViewModel associé, respectivement « MainViewModel » et « PopupViewModel » (MainViewModel n'étant pas indispensable pour l'exemple). L'apparition de la popup dans MainView est régie par les VisualStates « PopupOpennedVisualState » et PopupClosedVisualState ». La commande associée au bouton « Ok » va sauvegarder les informations contenues dans PopupViewModel et doit ensuite provoquer la fermeture de la popup. Cependant, étant donné que la sauvegarde est réalisée dans le ViewModel de la popup et que les Etats visuels sont définis dans la vue MainView, il y a ici deux problèmes :

- Hiérarchiquement, la popup est contenue dans la MainView, la popup n'a donc en théorie pas accès aux donnés de MainView.
- La confirmation de l'enregistrement se trouve au niveau des ViewModels alors que les états visuels sont au niveau de la vue. Là encore, théoriquement, la couche des ViewModels ne doit pas avoir accès à des informations telles que les états visuels qui sont des informations visuelles.

La TriggerAction « GotoStateAction » permet de résoudre ce dilemme facilement :

- Dans la classe PopupViewModel, Ajoutons un évènement « OnSaved » qui sera propagé lorsque les données de la Popup seront sauvegardées :

```
public class PopupViewModel
{
        […]

          private void Save
          {
              // TODO: Implémenter l'action d'enregistrement

          if(OnSaved != null)
          {
                OnSaved(this, new EventArgs()) ;
          }
          }

          public event EventHandler OnSaved;

        […]
}
```

- Dans la vue MainView, attachons à présent via Expression Blend notre TriggerAction « GotoStateAction » sur la Grid à la racine de la vue : il suffit de glisser/déposer la TriggerAction sur la Grid « LayoutRoot ». La TriggerAction apparaît ensuite juste sous la Grid indiquant qu'elle est bien attachée [Fig.3].
- Dans la fenêtre de propriété de la TriggerAction, il faut sélectionner la source des évènements comme étant le DataContext de la PopupView. Il suffit de cliquer sur le petit rectangle à droite du champ « Source object » (un menu apparaît) [Fig.4].

« Element Property Binding…» nous permet ensuite de sélectionner la popup, puis la propriété « DataContext » via la boîte de dialogue qui s'ouvre [Fig.5]. Nous pouvons à présent sélectionner l'évènement « OnSaved » dans la liste du champ « Event name » [Fig.6]. Reste enfin le nom de l'état visuel à sélectionner, et nous en avons fini avec le paramétrage de cette TriggerAction : [Fig.7]. En lançant l'application, nous constatons qu'en cliquant sur le bouton « Ok » la Popup se ferme effectivement. Créer un bouton permettant d'ouvrir la Popup depuis MainView est enfantin puisqu'il suffit d'attacher une GotoStateAction au bouton en sélectionnant l'évènement « click » et le VisualState « PopupOpenned ». De même en ce qui concerne l'état visuel par défaut (« PopupOpenned » ou « PopupClosed ») de la vue MainView, il suffit d'ajouter une GotoStateAction à la Grid à la racine de la MainView déclenchée sur l'évènement « Loaded » de la Grid.

CRÉATION D'UNE TRIGGERACTION SPÉCIFIQUE

Pour illustrer la création d'une TriggerAction spécifique, nous prendrons le cas de la gestion de boîte de dialogue en WPF. Le principe est simple : La plupart du temps pour afficher une boîte de dialogue (MessageBox, OpenFileDialog ou boîte de dialogue spécifique), le développeur n'a pas d'autre choix que d'implémenter cette gestion de la boîte de dialogue dans la partie « code-behind » ou ViewModel. Bien qu'il n'y ait pas de problème en soi, cette approche limite l'implémentation de tests unitaires ou encore l'objectif d'avoir un code indépendant des couches visuelles de l'application. En utilisant la mécanique d'une TriggerAction, nous allons montrer comment isoler le code de gestion d'une boîte de dialogue de type OpenFileDialog afin de rendre notre code du ViewModel indépendant des couches graphiques. Pour implémenter cette TriggerAction, nous commençons par créer un projet de type librairie de classes pour WPF 4 dans Visual Studio 2010. Puis nous ajoutons une référence à la librairie System.Windows.Interactivity.dll.

Le chemin d'accès dépend de l'installation du SDK d'Expression Studio et de la version de Blend ; Par exemple : C:\Program Files\MicrosoftSDKs\Expression\Blend\Silverlight\v4.0 \Libraries\ pour Silverlight, ou C:\Program Files\Microsoft SDKs\Expression\Blend\.NETFramework\v4.0\Libraries\System.Windows.Interactivity.dl pour WPF. Il est ensuite nécessaire d'ajouter une classe qui dérive de la classe TriggerAction :

```
using System;
using System.Windows;
```

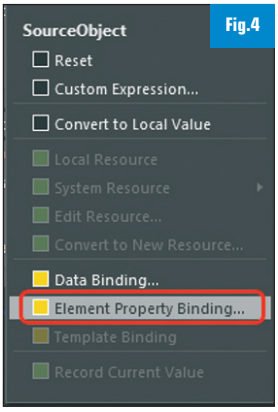


Fig.4

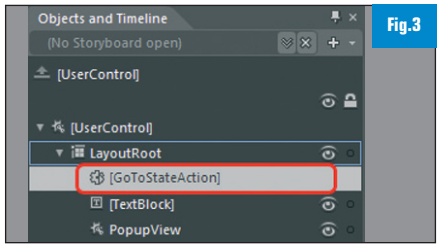


Fig.3

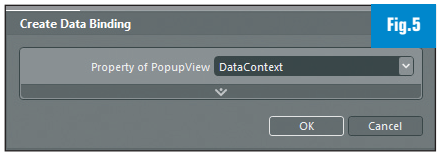


Fig.5

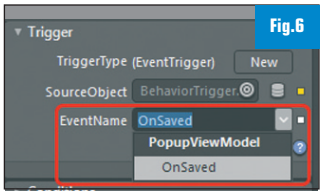


Fig.6

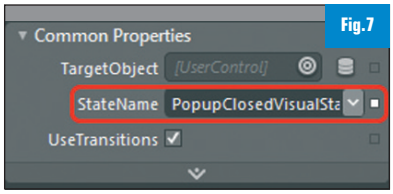


Fig.7

```
using System.Windows.Interactivity;
using System.Windows.Input;
using Microsoft.Win32;

namespace Samples.TriggerAction
{
    public class OpenFileDialogTriggerAction : TriggerAction
<UIElement>
    {
    }
}
```

Le type d'objet associé est UIElement car cette TriggerAction peut aussi bien être utilisée sur un bouton qu'un champ texte ou autre.
Le corps d'une TriggerAction réside dans l'implémentation de la méthode Invoke. Cette méthode est exécutée lorsque l'évènement auquel est lié la TriggerAction est déclenché. Ci-dessous l'exemple en code :

```
protected override void Invoke(object parameter)
{
    OpenFileDialog dialog = new OpenFileDialog();
    dialog.Filter = Filter;
    dialog.Title = Title;
    if (dialog.ShowDialog() == true)
    {
        // TODO: Implémenter quand l'utilisateur clique «OK»
    }
    else
    {
      // TODO: Implémenter quand l'utilisateur clique «Cancel»
    }
}
```

Le lecteur aura remarqué l'affectation des propriétés de la boîte de dialogue à partir de propriétés publiques de notre TriggerAction. Ces propriétés sont implémentées sous forme de DependencyProperty de la manière suivante :

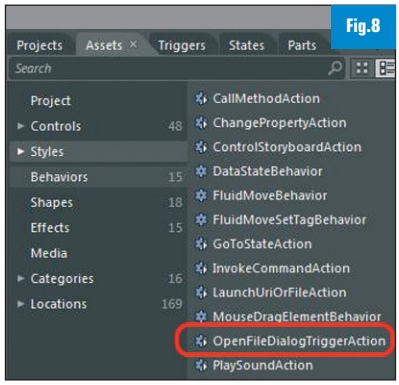
```
public static readonly DependencyProperty FilterProperty =
   DependencyProperty.Register(«Filter», typeof(string), type
of(OpenFileDialogTriggerAction), new UIPropertyMetadata(null));

public string Filter
{
    get { return (string)GetValue(FilterProperty); }
    set { SetValue(FilterProperty, value); }
}

public static readonly DependencyProperty TitleProperty =
    DependencyProperty.Register(«Title», typeof(string), type
of(OpenFileDialogTriggerAction), new UIPropertyMetadata(null));

public string Title
{
    get { return (string)GetValue(TitleProperty); }
    set { SetValue(TitleProperty, value); }
}
```

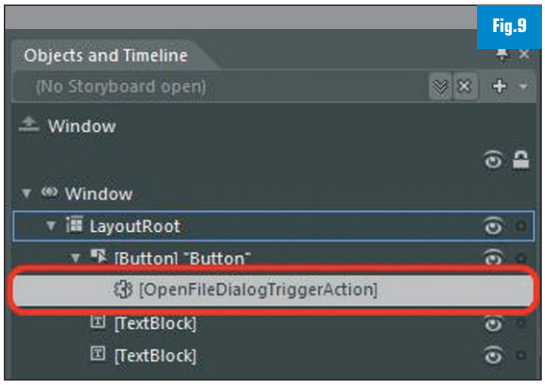
Enfin, il reste à implémenter deux propriétés permettant de référencer des commandes à exécuter lorsque l'utilisateur valide ou non le fichier à ouvrir. Le référencement des commandes (interface ICommand) est réalisé là aussi avec des DependencyProperties :

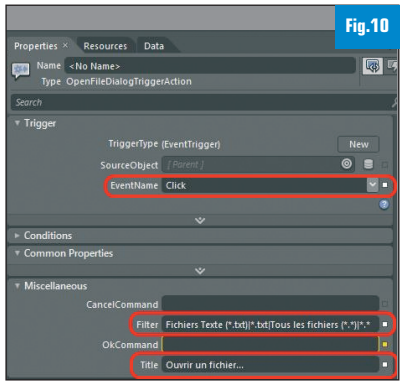
```
public static readonly DependencyProperty OkCommandProperty =
   DependencyProperty.Register(«OkCommand», typeof(ICommand),
typeof(OpenFileDialogTriggerAction), new UIPropertyMetadata(null));

public ICommand OkCommand
{
    get { return (ICommand)GetValue(OkCommandProperty); }
    set { SetValue(OkCommandProperty, value); }
}

public static readonly DependencyProperty CancelCommandProperty =
   DependencyProperty.Register(«CancelCommand», typeof(ICommand),
 typeof(OpenFileDialogTriggerAction), new UIPropertyMetadata
(null));

public ICommand CancelCommand
{
    get { return (ICommand)GetValue(CancelCommandProperty); }
    set { SetValue(CancelCommandProperty, value); }
}
```

Le code de la méthode Invoke finalisé :

```
protected override void Invoke(object parameter)
{
    OpenFileDialog dialog = new OpenFileDialog();
    dialog.Filter = Filter;
    dialog.Title = Title;
    if (dialog.ShowDialog() == true)
    {
```







```
        if ((OkCommand != null) && (OkCommand.CanExecute(dialog.
FileName)))
        {
            OkCommand.Execute(dialog.FileName);
        }
    }
    else
    {
      if ((CancelCommand != null) && (CancelCommand.CanExecute
(null)))
        {
            CancelCommand.Execute(null);
        }
    }
}
```

Dans le code ci-dessous, lorsque l'utilisateur valide l'ouverture de fichier, la commande « OkCommand » est exécutée avec le chemin d'accès au fichier passé en paramètre. Pour utiliser notre TriggerAction spécifique, nous implémentons une application WPF exemple dans laquelle nous positionnerons des contrôles et notamment un bouton qui déclenchera la TriggerAction sur un clic souris. Cette application exemple est implémentée selon le pattern M-V-VM afin d'illustrer la séparation du code du ViewModel de la partie visuelle. Afin que notre TriggerAction apparaisse dans la liste des Behaviors mis à disposition par Blend, il est nécessaire de référencer notre assembly précédemment compilée [Fig.8]. Pour l'appliquer sur un contrôle, il faut glisser/déplacer la Trigger-Action spécifique sur le contrôle ; Ici, un bouton : [Fig.9]. L'événement peut être sélectionné (ici, le Click) et les propriétés peuvent ensuite être modifiées : [Fig.10].

```
Ou le XAML peut être édité directement :
<Button Content=»Button» Grid.Row=»0»>
    <i:Interaction.Triggers>
        <i:EventTrigger EventName=»Click»>
            <my:OpenFileDialogTriggerAction
                Title=»Ouvrir un fichier...»
               Filter=»Fichiers Texte (*.txt)|*.txt|Tous les
fichiers (*.*)|*.*»
                />
        </i:EventTrigger>
    </i:Interaction.Triggers>
</Button>
```

La déclaration du namespace "my" est la suivante :

```
xmlns:my=»clr-namespace:Samples.TriggerAction;assembly=Samples.
TriggerAction»
```

Pour référencer le code à exécuter lors de l'ouverture du fichier, nous implémentons un ViewModel (nommé MainViewModel) qui expose une commande OpenFileCommand :

```
public ICommand OpenFileCommand { get { return new Delegate
Command(OpenFile); } }

public void OpenFile(object parameter)
{
    Filename = (string)parameter;
    FileContent = File.ReadAllText(Filename);
}
```

Les propriétés Filename et FileContent sont des propriétés publiques de la classe MainViewModel qui implémente l'interface INotifyPropertyChanged. Ces propriétés peuvent être « bindées » à des propriétés de contrôles pour afficher le fichier ainsi que son contenu. Le lecteur aura remarqué que le chemin d'accès au fichier à ouvrir est passé en paramètre de l'appel à la méthode OpenFile.
L'exemple ci-dessous pour la propriété FileContent peut être reproduit pour la propriété Filename :

```
private string _fileContent;

public string FileContent
{
    get { return _fileContent; }
    set
    {
        _fileContent = value;
        OnPropertyChanged(«FileContent»);
    }
}
```

Pour lier la commande OpenFileCommand au clic du bouton, nous déclarons le ViewModel et le référençons dans le DataContext de la grille principale :

```
<Window.Resources>
        <vm:MainViewModel x:Key=»MainViewModel»/>
</Window.Resources>

<Grid x:Name=»LayoutRoot» DataContext=»{Binding Source={Static
Resource MainViewModel}}»>
```

Nous pouvons alors référencer la commande depuis la TriggerAction spécifique. Le code XAML complet permet à l'utilisateur d'afficher le contenu d'un fichier texte sélectionné à partir d'une boîte de dialogue, elle-même ouverte lors du clic sur le bouton.

CONCLUSION

Dans cet article, nous avons introduit les mécanismes Behavior et TriggerAction offerts par Blend. Ces mécanismes permettent d'étendre les capacités des contrôles des applications WPF ou Silverlight tout en respectant les principes de séparation des couches applicatives. Bien que Blend intègre nativement les Behaviors et TriggerActions les plus couramment utilisés (CallMethod, InvokeCommandAction, MouseDragElementBehavior…), les développeurs ont la possibilité d'implémenter leurs propres comportements en dérivant des classes présentées dans l'article.

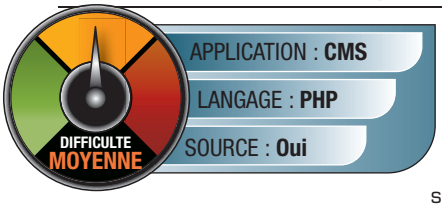
Références

Introduction aux Behavior et TriggerAction sur le blog de l'équipe Expression : http://blogs.msdn.com/b/expression/
Blog de Charles Hétier : http://charly-studio.com/blog/
La galerie Expression qui permet de télécharger des Behaviors réutilisables et développés par la communauté : http://gallery.expression.microsoft.com/

■ Charles **Hétier**
Ingénieur R&D – Orange Business Services.
■ Frédéric **Queudret**
CEO/CTO – MPoware. MPoware est une société française d'édition de logiciels et de prestations de services sur les technologies .NET.
http://www.mpoware.com/

Drupal 6 : personnaliser le module de e-commerce Ubercart

Module de e-commerce pour Drupal, Ubercart est un véritable système dans le système. Doté à la base de riches fonctionnalités, il est extensible et personnalisable. Nous découvrons aujourd'hui comment procéder.



Drupal 6 est un gestionnaire de contenu écrit en PHP et extensible. Les fonctionnalités de base qu'il propose sont complètes, dès lors qu'il s'agit de contenus basiques comme un billet de blog. Tôt ou tard, vient le jour où il est nécessaire de le doter de fonctionnalités plus spécifiques. On écrira alors un module d'extension à partir de son API. Il est très fréquent qu'un site doive être doté de fonctionnalités de e-commerce: panier, gestion des commandes, des factures, du stock, etc. Il existe plusieurs modules d'extensions de e-commerce pour Drupal, dont le plus complet et le plus utilisé est probablement Ubercart (ubercart.org). Ce module est open source et il est remarquablement complet. Cependant, tôt ou tard, ses fonctionnalités devront être adaptées à la spécificité d'un site. Véritable système dans le système, Ubercart expose à son tour une API qui permet de lui ajouter des extensions. De fait, de nombreux modules d'extensions à Ubercart existent déjà, dont certains exposent eux-mêmes une API, et il est bien entendu possible d'en écrire d'autres. Tout cet ensemble peut paraître bien compliqué, mais pas tant que cela, dès que l'on a acquis quelques bases, que cet article va s'efforcer de donner.

1 CAHIER DES CHARGES ET OUTILS

Nous voulons que notre site facilite la vie du visiteur selon que celui-ci est un particulier ou le responsable des achats d'une entreprise. Concrètement, nous voulons proposer au visiteur le moyen de basculer l'affichage des prix T.T.C ou H.T. Les prix doivent alors être recalculés à la volée, et l'affichage modifié partout: catalogue, panier, bon de commande, etc. L'administrateur doit avoir la possibilité de modifier le taux de taxe, et le site doit se rappeler du choix du visiteur, afin que celui-ci, s'il revient sur le site, ne soit pas contraint de redemander un changement d'affichage. Nous allons travailler avec Drupal 6 et Ubercart 2.0. Contrairement à certains articles relatifs à Drupal précédemment publiés, nous ne parlerons pas de Drupal 7, car Ubercart n'est pas encore porté pour cette version. Nous supposons que le lecteur est à l'aise avec l'interface d'administration de Drupal. Nous supposons aussi que le lecteur possède quelques connaissances de base en ce qui concerne la programmation des modules pour Drupal. A ce titre, il peut être pertinent de (re)lire *Ecrire un module d'extension pour Drupal* paru dans Programmez! 123 et *Intégrer Paypal à Drupal 6* paru dans Programmez! 144. Ce dernier article n'étant pas dédié à Paypal proprement dit, mais bien à l'étude de points d'intégration dans Drupal 6.

2 EXPLOITER LA DOCUMENTATION

Le lecteur qui souhaiterait approfondir cet article et l'adapter à ses besoins trouvera toute la documentation relative à Drupal à drupal.org. Il trouvera la documentation de l'API Ubercart à ubercart.org. Dans cette partie de la documentation, les informations relatives à l'API Price sont manquantes. On trouvera celles-ci à Developer's Guide -> Developing for Ubercart 2.x -> Core systems. Enfin Ubercart vient avec un fichier hooks.php dans le sous-répertoire docs de son archive. Ce fichier, comme son nom ne l'indique pas, est une documentation des hooks d'Ubercart, et non un fichier destiné à être exécuté.

3 LES HOOKS DE DRUPAL ET D'UBERCART

Nous savons que Drupal est construit sur un système de hooks. Il s'agit de fonctions de rappel qui doivent obéir à une convention de nommage, et que le système va se charger d'invoquer. Si Drupal peut invoquer un hook, un programmeur peut le faire aussi. C'est le principe de Ubercart. Quand Drupal invoque les hooks 'Drupal' de Ubercart, celui-ci peut alors à son tour invoquer des fonctions de rappel obéissant à une convention de nommage: les hooks Ubercart. Cette plomberie est transparente au programmeur et si celui-ci sait écrire un module pour Drupal, il sait aussi, potentiellement, écrire une extension à Ubercart.

4 LES POINTS D'INTÉGRATION

En ce qui concerne Drupal, Il faut tout d'abord un formulaire dans l'interface d'administration, afin que l'administrateur du site puisse régler le montant de la taxe. Nous avons ensuite besoin d'URL de rappel pour réagir à la demande de l'utilisateur de basculer l'affichage. Ces URL seront définies via des entrées de menu non affichées. Des liens pointant sur ces URL seront présentés à l'utilisateur dans un bloc. Enfin les données utilisateur, ou la session pour les utilisateurs anonymes permettront de conserver la configuration du site. En ce qui concerne Ubercart, nous devons signaler notre gestionnaire de prix, et en écrire le code.

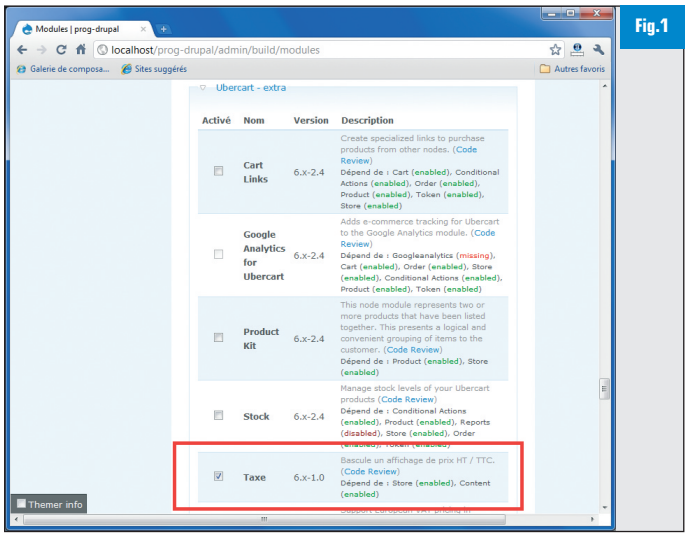


Fig.1

Grâce à notre fichier .info notre module apparaît listé parmi les suppléments à Ubercart.

5 MISE EN PLACE

Partons du principe que nous disposons d'un Drupal 6 installé, avec Ubercart 2.0 installé dans celui-ci, avec ses fonctionnalités classiques (Panier, catalogue, commande, etc.) activées. Nous décidons de vendre des compilateurs présentés dans le catalogue, et les classes de produits correspondant sont définies. Par convention, les prix saisis dans l'interface du magasin sont hors taxes. Notre module, même s'il étend Ubercart est un module Drupal classique. Commençons par écrire son fichier de définition (uc_taxes.info)

```
name = Taxe
description = "Bascule un affichage de prix HT / TTC."
dependencies[] = uc_store
dependencies[] = content
package = "Ubercart - extra"
core = 6.x
version = "6.x-1.0"
```

Nous baptisons notre module Taxe, à ne pas confondre avec le module Ubercart Taxes. Notre module dépendra des modules Content (Drupal) et Store (Ubercart). Enfin nous plaçons notre module dans un package, afin que l'interface d'administration des modules le présente dans un endroit conforme au bon sens [Fig.1].

6 CONFIGURATION DU MODULE

Nous devons présenter un formulaire à l'administrateur du site, afin que celui-ci puisse saisir la valeur de la taxe. Drupal simplifie la création des formulaires. Il suffit de fournir un tableau à clés et le système construira le formulaire automatiquement [Fig.2].

```
<?php
/**
 * Definit le formulaire de configuration du module
 */
function uc_taxe_form_settings() {
 $form['uc_taxe_configuration'] = array(
   '#type' => 'textfield',
   '#title' => t('Taux de la taxe à appliquer sur les prix'),
   '#default_value' => variable_get('uc_taxe_configuration', '1'),
   '#size' => 10,
   '#maxlength' => 10,
   '#description' => t(«Donner une nombre»),
   '#required' => TRUE,
  );

  return system_settings_form($form);
}
```

Pour une fois, ce code n'est pas un hook :-) Il s'agit d'une simple fonction qui sera appelée lorsque l'administrateur cliquera sur l'entrée de menu correspondante. Entrée qui, logiquement, doit faire partie du menu de configuration de la boutique. Nous verrons plus loin comment obtenir ceci. Bien remarquer dans ce code l'appel à la fonction système Drupal variable_get, qui remplit le champ du formulaire avec une valeur saisie précédemment, ou 1, si rien n'a encore été saisi. Lorsque l'administrateur valide ainsi un formulaire de configuration, le système sauvegarde automatiquement les données dans une table système qui, curieusement, est baptisée 'variable' et non variables. La fonction variable_get sait extraire la donnée de la table. Ce mécanisme n'est bien sûr valable que pour les petits volumes de données, telles que les variables de configuration. Enfin, notre code est placé dans un fichier uc_taxe.admin.inc et non dans le fichier uc_taxe.module qui contient le gros du code de notre module. L'intention est d'économiser des ressources, Drupal ne chargeant alors le code que lors des visites à l'interface d'administration. Drupal est informé du nom de fichier par la définition du menu que nous verrons plus loin.

7 DÉFINIR UN GESTIONNAIRE DE PRIX

Nous entrons maintenant dans le domaine de Ubercart. Celui-ci prévoit que les prix d'un produit puissent être modifiés à la volée par un ou plusieurs gestionnaires de prix (ou price_handler). Ceux-ci sont de simples fonctions php qui sont invoquées dans un ordre déterminé. L'ensemble définit un pipeline. L'administrateur définit l'ordre des gestionnaires depuis l'administration de la boutique et doit être conscient que l'ordre des gestionnaires influe sur le résultat final [Fig.3]. La déclaration d'un gestionnaire de prix dans le code n'en définit pas le positionnement dans le pipeline. Cette déclaration se fait dans un hook :

```
/**
 * Implementation de hook_uc_price_handler()
 * Voir API Price de Ubercart
 */
function uc_taxe_uc_price_handler() {
```

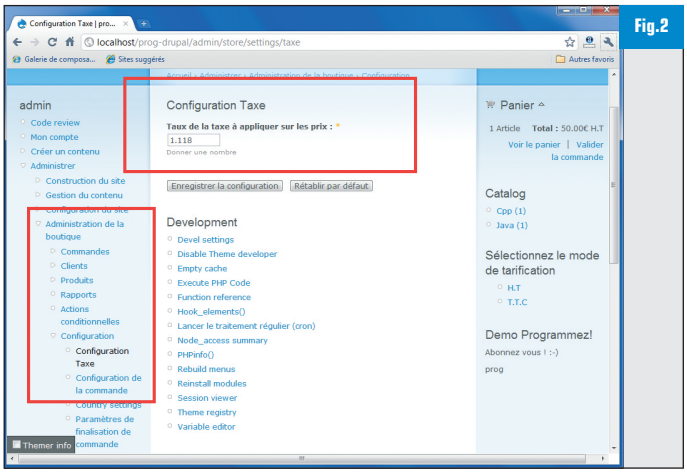


Fig.2

Notre module présente un formulaire de saisie du taux de taxe à l'administrateur du site.

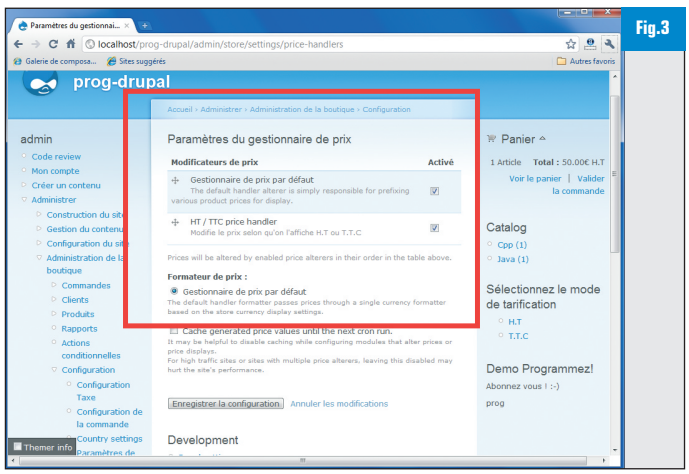


Fig.3

A charge de l'administrateur du site d'ordonner correctement les gestionnaires de prix.

```
  return array(
    'alter' => array(
    'title' => t('HT / TTC  price handler'),
    'description' => t(«Modifie le prix selon qu'on l'affiche H.T ou T.T.C»),
    'callback' => 'uc_taxe_price_handler_alter',
    ),
  );
}
```

Classiquement nous retournons un dictionnaire à clé. La clé 'alter' signifie que notre gestionnaire modifiera le montant du prix. Nous aurions pu donner à la place ou en plus 'format' pour redéfinir le format du prix, ou 'options' pour redéfinir le symbole de la devise. Et toujours classiquement avec Drupal/Ubercart, notre tableau définit une fonction de rappel, ici uc_taxe_price_hanlder_alter.

8 L'API PRICE DE UBERCART

Un des points délicats de notre travail est de comprendre le fonctionnement de l'API Price de Ubercart, qui n'est pas clairement documentée (http://www.ubercart.org/docs/developer/11375/price_api). Selon la nature de notre fonction de rappel, celle-ci recevra 2 ou trois arguments. Pour nous qui voulons modifier un prix, c'est trois arguments. Le premier est un tableau à clés. Le premier élément de ce tableau est le prix du produit que traite le gestionnaire. Le second élément est une quantité qui vaut toujours au moins un et dont la valeur exacte dépend de ce dont on parle. Si c'est d'un produit sans gestion de stock, la quantité est un. Si c'est d'un produit dans le panier, la quantité est le nombre de ce produit dans le panier. Si c'est le montant total du panier, la quantité est un, quel que soit le nombre d'éléments dans le panier. Difficile de s'y retrouver à partir de cette seule information… C'est pourquoi le second argument reçu par la fonction décrit un contexte. Celui-ci est constitué à nouveau d'un tableau à clés. L'élément fondamental de ce tableau est le type (amount, product, cart_item, etc.). Selon ce type, notre tableau, sous la clé 'subject' contiendra un autre tableau à clés. Le contenu de ce second tableau est détaillé dans la documentation qui doit maintenant paraître plus claire au lecteur. Un bon moyen de bien comprendre est de définir notre fonction ainsi :

```
function uc_taxe_price_handler_alter(&$price, &$context, &$options) {
  echo '<pre>';
  echo '<p>' . print_r($price) . '</p>';
  echo '<p>' . 'end price' . '</p>';
  echo '<p>' . print_r($context) . '</p>';
  echo '<p>' . 'end context' . '</p>';
  echo '<p>' . print_r($options) . '</p>';
  echo '<p>' . 'end options' . '</p>';
  echo '</pre>';
}
```

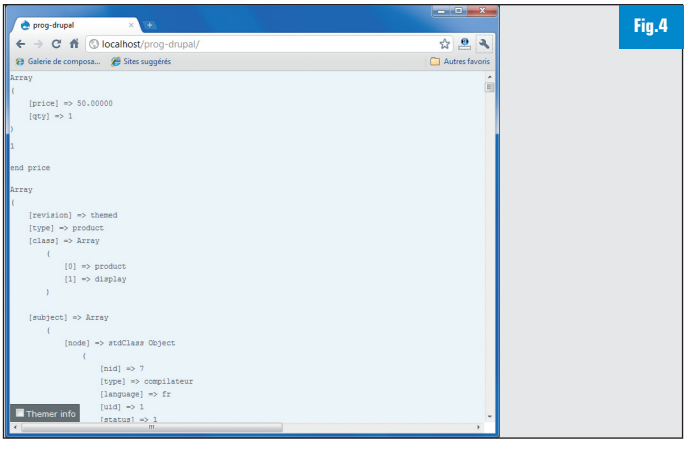


Fig.4

N'hésitez pas à tracer le comportement d'un gestionnaire de prix pour bien comprendre le fonctionnement d'Ubercart.

L'affichage produit dans la page (ou dans la source de la page) [Fig.4], permet de mieux s'y retrouver, et surtout de constater que notre gestionnaire de prix peut être invoqué un très grand nombre de fois pour une seule page. Le dernier argument reçu par la fonction concerne le signe de la devise, et il ne présente pas de difficulté. Voici maintenant le code réel de notre gestionnaire de prix :

```
function uc_taxe_price_handler_alter(&$price, &$context, &$options) {
  // Récupérer le mode de tarification
  global $user;
  $curmode =  'H.T'; // par défaut
  if ($user->uid && !empty($user->uc_taxe)) {
      $curmode = $user->uc_taxe;
  }
  elseif (!empty($_SESSION['uc_taxe'])) {
      $curmode = $_SESSION['uc_taxe'];
  }

  // Récupérer le taux à appliquer
  $taux = variable_get('uc_taxe_configuration', '1');
  //echo $curmode;
  switch ($context['type']) {
    case 'product':
    case 'cart_item':
    case 'order_product':
      $node = $context['subject']['node'];
      // S'assurer que les noeuds sont complètement chargés
      // Ce n'est peut être pas le cas dans le cas des vues
      if (!isset($node->type) || !isset($node->sell_price) || !isset
($node->shippable))  {
        $node = node_load($node->nid);
      }
      if($curmode == 'T.T.C') {
        $price['price'] = $price['price'] * $taux;
      }
      break;

  }
  // ATTENTION ci-dessous MAUVAIS ENDROIT pour modifier
  // prix, car la modification sera faite en sus pour d'autres
types de contexte
  // tels que payment...
  //$price['price'] = $price['price'] * etc

  // Modifier l'affichage est ici CORRECT OU PAS
  // selon le fonctionnement global du site. Voir
  // remarque ci-dessus.
  $options['sign'] = $options['sign'] . « « . $curmode;
}
```

Deux remarques très importantes. D'abord notre gestionnaire peut-être invoqué moult fois à des moments inattendus et pour des

contextes variés. C'est pourquoi il est capital de toujours bien tester le contexte et de n'agir que dans le cadre de celui-ci, sinon des modifications de prix peuvent être malencontreusement cumulées. Enfin dans certains cas, comme par exemple l'affichage de vues par champs, il n'est pas garanti qu'un nœud de produit soit complètement chargé. D'où un appel à node_load dans notre code. Enfin, dans ce gestionnaire, nous lisons la configuration de l'utilisateur, soit dans son compte soit dans une variable de session, afin de produire l'affichage souhaité.

9 LE MENU

Voici maintenant comment définir le menu pour notre module, en implémentant hook_menu :

```
/**
 * Implementation de hook_menu
 */
function uc_taxe_menu() {

 $items = array();
 $items['admin/store/settings/taxe'] = array(
   'title' => 'Configuration Taxe',
   'description' => «La configuration permet de définir la taxe
applicable aux prix»,
   'page callback' => 'drupal_get_form',
   'page arguments' => array('uc_taxe_form_settings'),
   'access arguments' => array('access administration pages'),
   'type' => MENU_NORMAL_ITEM,
   'file' => 'uc_taxe.admin.inc',
 );

 $items['uc_taxe_set/%'] = array(
     'title' => 'Selection du mode de taxe',
     'access arguments' => TRUE,
     'page callback' => 'uc_taxe_set_callback',
   'access callback' => TRUE,
     'page arguments' => array(1),
     'type' => MENU_CALLBACK,
 );
 return $items;
}
```

D'abord nous définissons le menu pour l'administration. Nous donnons le chemin qui fera que le menu apparaîtra au bon endroit, et nous disons au système dans quel fichier se situe le code. Ensuite nous définissons un menu invisible, ou simple fonction de rappel (MENU_CALLBACK). Le % du chemin 'uc_taxe_set/%' signifie que nous passerons des arguments. Le nombre d'arguments, ici 1, est donné par la taille du tableau 'page arguments'. Ce nombre doit être exact. Par exemple array(2) ne fonctionnerait pas avec notre exemple.

10 INVOQUER LA FONCTION DE RAPPEL

Ceci peut se faire, par exemple, à partir de liens présentés dans un bloc PHP, dont le code pourrait être :

```
<?php

// ATTENTION, adaptez le chemin à votre configuration
echo '<ul>';
echo '<li>';
echo '<a href=»/prog-drupal/uc_taxe_set/ht?destination=' . $_GET
['q'] . '«>H.T</a>';
echo '</li>';
echo '<li>';
echo '<a href=»/prog-drupal/uc_taxe_set/ttc?destination=' . $_GET
['q'] . '«>T.T.C</a>';
echo '</li>';
echo '</ul>';

?>
```

Bien comprendre ici que l'unique argument attendu par le callback de menu est ht ou ttc et qu'il fait partie intégrante du chemin. L'autre argument est transmis classiquement dans une requête GET. Nous passons la page en cours, afin de rediriger vers cette page, autrement dit, nous rafraîchissons la page.

11 CONSERVER LA CONFIGURATION DE L'UTILISATEUR

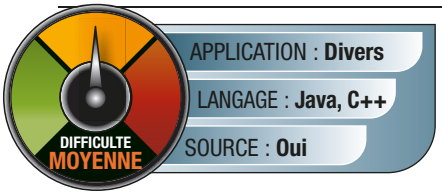
Voici finalement le code de notre fonction de rappel. Nous y sauvegardons le choix de l'utilisateur, dans son compte, s'il s'agit d'un utilisateur enregistré, dans une variable de session si ce n'est pas le cas.

```
function uc_taxe_set_callback($mode) {
    global $user;

    switch($mode) {

     case 'ttc':
       $curmode = 'T.T.C';
       break;
     case 'ht':
       $curmode = 'H.T';
       break;
     default:
       $curmode = 'H.T';
       break;
    }
    if ($user->uid) {
     // L'utilisateur est authentifié
     user_save($user, array('uc_taxe' => $curmode));
    }
    else {
     // L'utilisateur est anonyme
     if (!empty($mode)) {
       $_SESSION['uc_taxe'] = $curmode;
     }
     else {
       unset($_SESSION['uc_taxe']);
     }
    }

    $dest = $_GET['destination'];
    drupal_goto($dest);
}
```

Frédéric **Mazué** - *fmazue@programmez.com*

Exceptions : quelques bonnes pratiques en Java et C++

Si les exceptions sont un véritable progrès dans l'écriture du code, elles ne sont en aucun cas synonymes ni de facilité, ni de robustesse.



A l'époque de la préhistoire, c'est-à-dire il y a quelques décennies à peine, l'instruction goto sévissait cruellement et les programmes spaghetti, inextricables, totalement incompréhensibles et illisibles, souvent écrits par des scientifiques plutôt que des informaticiens, étaient partout. Puis, il fut démontré qu'il est possible d'écrire un programme sans aucun goto et globalement le code se trouva assaini, écrit plus vite, et avec plus de sûreté. Néanmoins, le code constitué de tests imbriqués avec des codes d'erreur qui doivent remonter plusieurs appels de fonction, devient vite un pensum lui aussi, long à concevoir et à mettre au point. C'est ici qu'arrivent les exceptions. Celles-ci sont un véritable progrès, car elles permettent d'écrire le code plus vite, et avec plus de clarté, en faisant facilement remonter une condition d'erreur à toute une pile d'appels. Tel est leur bon aspect. Mais des exceptions utilisées avec insouciance, par paresse ou facilité peuvent vite devenir de fausses amies, génératrices de problèmes subtils. Ecrire du code (qui marche :) a toujours été et restera difficile. En l'occurrence, parce qu'elles brisent le flux d'exécution, les exceptions ont un petit quelque chose du goto de la préhistoire, qui, masqué, vient prendre sa revanche. Essayons de le déjouer. Commençons avec Java.

1 NE JAMAIS RETOURNER DE VALEUR DEPUIS UN BLOC TRY

Parce qu'elles brisent le flux d'exécution, les exceptions ne font pas bon ménage avec les instructions *return*. Voyons ce code :

```
boolean noTryReturn() {
  try {
    int i = 1/0;
    return true;
  }
  catch(Exception e) {
    return true;
  }
  finally {
    return false;
  }
}
```

Nous avons placé une instruction return dans un bloc try, ce qui est une grosse faute, même si le compilateur reste totalement muet. Si nous sommes dans un bloc try, c'est que le code est susceptible de lever une exception. Qu'une valeur soit à coup sûr retournée par ce bloc est donc totalement aléatoire. Ici nous levons effectivement une exception par une division par zéro. Mais même si nous supprimons la ligne de code qui lève l'exception, notre méthode ne retourne jamais *true*. En effet Java certifie que le bloc *finally* sera exécuté quoi qu'il arrive (sauf bien sûr dans le cas du crash de la JVM ou d'un appel à System.exit(), mais dans de tels cas, la discussion n'a plus de sens de toute façon). Il est alors tentant de supprimer le bloc *finally* :

```
public boolean noTryReturn() {
  try {
    int i = 1/0;
    return true;
  }
  catch(Exception e) {
    // traitement de l'exception.
    return true;
  }
}
```

Mais c'est une erreur. D'abord c'est laid, parce qu'il y a deux instructions *return* pour, finalement, une seule condition de départ. Ensuite, ce code est plus difficile à maintenir, et il est faussement sécurisant. En effet il est possible que le traitement lui-même de l'exception, dans le bloc *catch* lève à son tour une exception, et notre méthode n'ira pas à son terme. Or ce n'est pas a priori ce que nous attendons d'elle. Le code correct est donc :

```
public boolean noTryReturn() {
  try {
    int i = 1/0;
  }
  catch(Exception e) {
    // traitement de l'exception.
  }
  finally {
    // selon la situation.
    return true; // ou false
  }
}
```

Ici, si le bloc *catch* lève une exception, le bloc *finally* sera malgré tout exécuté et notre méthode aura le comportement attendu: retourner une valeur. Mais attention: il faut être conscient qu'une éventuelle exception levée dans le bloc *catch* se retrouverait ainsi muselée. C'est le revers de la médaille. Une exception signalant une condition … exceptionnelle, il est peut-être fâcheux que le programme n'en soit pas informé. Enfin lorsqu'on écrit un tel code, il faut être conscient que le contenu du bloc *finally* doit être réduit au minimum, c'est-à-dire constitué d'un code simple qui ne risque pas de lever une

exception. Dans le cas contraire, nous devrions dire adieu à notre valeur de retour, et notre programme pourrait se retrouver contraint de faire face à une exception à laquelle il ne s'attend pas. Nous parlons bien sûr ici d'exceptions non contrôlées, par exemple celles issues de calculs numériques, qui sont donc dangereux dans un bloc *finally*. N'oublions surtout pas non plus que la très redoutable exception NullPointerException, bien connue de tous les programmeurs Java, est une exception non contrôlée. Concrètement cela veut dire que manipuler un objet (et a fortiori en créer un) dans un bloc *finally*, est susceptible de mettre à mal le fonctionnement d'un programme. L'erreur d'indice dans un tableau est également un piège fameux ici.

2 NE JAMAIS MUSELER UNE EXCEPTION

Si les exceptions non contrôlées sont susceptibles de créer bien des soucis, comme nous venons de le voir, les exceptions contrôlées sont en général très pénibles pour le programme, car le compilateur l'oblige à les traiter. Grande est alors la tentation de museler les exceptions comme ceci :

```
void lecture () {
 try {
     FileInputStream in = new FileInputStream("nom_de_fichier");
 } catch (Exception ex) {} // on fait rien ici !?
}
```

Bien évidemment ceci ne peut pas convenir, car si le programme rencontre une erreur, par exemple un fichier absent, il continuera malgré tout son cours comme si de rien n'était à cause de cette exception muselée. Souvent, on fait cela pendant la mise au point du programme avec l'intention de reprendre le code plus tard… puis on oublie. La bonne pratique est de propager l'exception. D'ailleurs les éditeurs modernes nous y aident en proposant d'ajouter la clause à notre code, comme ceci :

```
void lecture () throws FileNotFoundException {
 FileInputStream in = new FileInputStream("nom_de_fichier");
}
```

3 ATTENTION À LA GESTION DES RESSOURCES

Les exceptions mal gérées peuvent conduire à l'épuisement de ressources. Voici le cas très classique de la connexion à une base de données :

```
Connection connect =  null;
try
{
 Class.forName("le_driver");
 connect = DriverManager.getConnection( // etc);
 // faire quelque chose.
 connect.close();
}
catch (Exception e)
{
 // écrire dans un log
}
```

Si, pour une raison lambda, "faire quelque chose", lève une exception, la connexion ne sera pas fermée immédiatement par l'appel à *close* du bloc *try*, mais ne sera fermée que lorsque le ramasse-miettes détruira l'objet, c'est-à-dire à un moment imprévisible et, surtout, différé. Si notre application est très sollicitée et que des exceptions sont fréquemment levées, nous atteindrons le nombre de connexions maximal supporté par la base de données et l'application ne fonctionnera plus convenablement. Pour cette raison, toutes les ressources rares (connexions base de données, socket, etc.) doivent être fermées/libérées dans un bloc *finally* comme ceci :

```
Connection connect = null;
try
{
 Class.forName("le_driver");
 connect = DriverManager.getConnection( // etc. );
 // faire quelque chose.
}
catch (Exception e)
{
 // écrire dans un log
}
finally {
 if(connect != null) {
   try {
     connect.close();
   }
   catch (Exception e){
     // réagir
   }
 }
}
```

Et bien entendu le conseil vu précédemment s'applique ici. Aucun code dans le bloc finally ne doit lever d'exception non traitée en amont de la fermeture de la connexion. Ce point important de la gestion des ressources rares sera toutefois amélioré avec Java 7 et son Automatic Resource Management.

4 VEILLER AU BON ÉTAT DES OBJETS

Encore un point particulièrement délicat. Les exceptions peuvent laisser un objet dans un état instable ou incohérent. Supposons que nous écrivons une classe gérant un compte en banque. L'argent est en jeu, la plus grande rigueur s'impose :-) Notre classe, collaborant dans une application, utilise d'autres classes et est utilisée par d'autres classes. En aucun cas les données encapsulées par notre classe ne doivent être incohérentes. Voici un premier jet :

```
public class BadAccount {
 private int operations = 0;
 private double montant = 0.0;

 public BadAccount() {
 }

 public void update(double somme) {
   operations = new Incrementator().calcule(operations);
```

```
    // le Goto masqué nous menace ici :-(
    montant = new Creditor().calcule(montant, somme);
  }
}
```

Notre classe détient le nombre des opérations effectuées et le montant disponible sur le compte. Ces deux valeurs doivent toujours rester cohérentes. Pour l'exemple nous supposons que l'incrémentation du nombre d'opérations est délégué à une classe *Incrementator*, et la mise à jour du montant est déléguée à une classe Creditor. Nous supposons que ces classes doivent être instanciées au moment de leur utilisation Et bien ce code tout simple ne fonctionne tout simplement pas! Telle que nous présentons la chose, les méthodes "calcule" de *Incrementator* et *Creditor* ne lèvent aucune exception contrôlée.

Mais le problème est qu'elles peuvent toujours (insistons sur le toujours) lever une exception non contrôlée. Par exemple, l'exception non contrôlée OutOfMemoryError peut être levée. Si elle est levée à l'instanciation de Creditor, alors les valeurs d'opérations et de montants ne seront plus cohérentes. Si l'exception n'est jamais capturée la JVM s'arrête. Est-ce satisfaisant ? Non, car il est plus que probable que notre application capture l'exception en amont de notre code pour sérialiser et sauvegarder tous les comptes en banque. En ce cas, notre compte sera sauvegardé dans un état incohérent car notre code n'est pas robuste aux exceptions.

Pour remédier à cela, les méthodes doivent avoir un comportement dit atomique. Autrement dit une opération doit être faite complètement ou pas du tout. On appelle cela le "valider ou annuler", ou encore par analogie avec les bases de données, le *Commit/Rollback*. Pendant la phase de validation on ne doit exécuter que du code dont on est absolument sûr qu'il ne lèvera aucune exception. En Java, il s'agit de la manipulation de types primitifs (sauf pour la division par zéro…) et l'affectation. Bref, nous n'avons réellement que l'affectation. Mais c'est suffisant :)

En réécrivant notre méthode *update* ainsi :

```
public void update(double somme) {
  // on travaille avec des variables temporaires
  int operations_ = new Incrementator().calcule(operations);
  double montant_ = new Creditor().calcule(montant, somme);
  // on valide le travail
  operations = operations_;
  montant = montant_;
}
```

nous sommes certains que l'état de notre objet sera toujours cohérent. Certains ? Maintenant que nous avons compris le principe, nous pouvons l'avouer, notre code n'est pas encore très bon. En effet en situation réelle, notre classe sera plus complexe et nous ne pourrions sans doute pas être sûrs d'être à l'abri d'effets de bord se produisant dans le corps d'update.

Ou même, si nous devons écrire de nombreuses méthodes comme celle ci-dessus, nous sommes à la merci d'étourderies pouvant nous faire oublier de travailler avec une variable temporaire ici ou là. Le mieux est d'utiliser l'idiome dit de *swap*, très connu des programmeurs C++.

Nous n'utilisons plus de variables temporaires, mais carrément une copie temporaire de notre classe :

```
public class BetterAccount implements Cloneable {
  private int operations = 0;
  private double montant = 0.0;

  private void swap(BetterAccount temp) {
    this.operations = temp.operations;
    this.montant =  temp.montant;
  }

  public void update(double somme) {
    try {
      BetterAccount temp = (BetterAccount)this.clone();
      temp.operations =
        new Incrementator().calcule(operations);
      temp.montant =
        new Creditor().calcule(montant, somme);
      // valider
      swap(temp);
    } catch (CloneNotSupportedException ex) {
      // cette exception ne doit
      // normalement pas se produire
    }
  }
}
```

Ici nous pouvons nous contenter de la méthode *clone* par défaut. Mais si notre classe contenait des références d'objets alors nous devrions implémenter notre propre méthode *clone*. Dernière remarque, notez bien que *swap* est *private* et non pas *public*, *protected* ou sans qualificateur de visibilité (visibilité dans tout le package), afin d'éviter les traquenards inhérents au polymorphisme. Tout n'est pas dit, loin s'en faut, sur ce vaste et difficile sujet. Nous disposons cependant de bonnes pratiques pour éviter de mauvais tours qui peuvent être joués par les exceptions en Java.

5 EN C++, LES DESTRUCTEURS DE C++ NE DOIVENT JAMAIS LEVER D'EXCEPTIONS

Ce que nous disons maintenant s'appuie sur les recommandations formulées par Herb Sutter, expert C++, à la suite de la résolution du problème posé par Tom Cargill et concernant la robustesse du code C++ aux exceptions. Le sujet est vaste et est là encore un vrai défi au programmeur. Un bon code doit, comme en Java, assurer l'état cohérent des objets. Il doit aussi ne pas produire de fuites mémoires. Le programmeur C++ doit veiller à ce point mais aussi avoir à l'esprit que les classes qu'il écrit sont susceptibles d'être réutilisées, notamment dans des conteneurs génériques. En Java, les ressources rares sont libérées dans un bloc *finally*, et le reste est libéré par le ramasse-miettes, ainsi que nous l'avons vu. En C++, le bloc *finally* n'existe pas. En C++ on travaille avec le paradigme RAII (Resource Acquisition Is Initialization). Avec ce paradigme, une méthode spéciale, le destructeur est appelé lorsqu'un objet quitte une portée. Ceci présente le grand avantage sur Java d'éviter la fuite de ressources rares, comme nous l'avons vu plus haut. Mais l'inconvénient est que pour que ceci fonctionne, le flux d'exécution ne doit pas être cassé. Autrement dit, un destructeur C++ ne doit jamais lever d'exception. Soit par exemple un conteneur générique, une pile :

```
template <class T> class Stack
{
public:
 Stack();
 ~Stack();
 // etc.
private:
 T* v_; // pointeur sur la mémoire utilisée par le conteneur
 size_t vsize_; // taille totale
 size_t vused; // taille réellement utilisée
}
```

Probablement ce conteneur aura un constructeur par défaut comme ceci :

```
template <class T> Stack<T>::Stack()
: v_(0), vsize_(10), vused_(0)
{
 v_ = new T[vsize_];
}

ainsi qu'un destructeur:

template <class T> Stack<T>::~Stack()
{
 delete[] v_;
}
```

Et bien sûr le type T sera peut-être une classe écrite par nous. Alors, dès l'écriture nous devons penser à cette collaboration. Par défaut, le constructeur de notre Stack alloue une zone mémoire de 10 objets. Chacun des 10 objets sera construit par le constructeur par défaut de l'objet. Constructeur qui est susceptible de lever une exception. En outre, l'allocation mémoire pour les objets eux-mêmes est toujours susceptible de lever une exception bad_alloc. Que se passe-t-il si la construction d'un des objets contenus échoue en levant une exception ? Le destructeur de Stack sera automatiquement invoqué (sortie de portée). Ce destructeur invoque l'opérateur delete[] dont la norme nous garantit qu'il ne lève pas lui-même d'exception. Cet opérateur delete[] va se charger d'invoquer les destructeurs de chacun des objets T déjà construits, puis il libérera la zone mémoire de Stack. Tout ceci fonctionnera parfaitement… à la condition que le destructeur d'un des objets T ne lève une exception. Si c'était le cas, le flux d'exécution serait brisé, et les objets T en aval ne seraient pas correctement détruits.
La zone mémoire gérée par notre Stack ne serait pas libérée non plus (saut dans terminate()). Voilà pourquoi le destructeur d'une classe bien écrite ne doit en aucun cas lever d'exception. Si l'éventualité d'une exception est incontournable, alors son gestionnaire résidera dans le destructeur lui-même, afin que l'exception ne puisse s'en échapper.

6 ATTENTION AU CODE C++ APPAREMMENT ANODIN

Ce simple code de destructeur, par exemple, est très mauvais :

```
MaClasse:~MaClasse()
{
  cout << "MaClasse détruite" << endl;
}
```

car il est susceptible de lever 3 exceptions, au moins ! D'abord, comme le précise la norme, chacun des appels à operator<<() est susceptible de lever une exception. Cela nous en fait déjà deux… Ensuite le message "MaClasse détruite" peut être converti, selon l'implémentation, en un objet temporaire. Ceci peut aboutir au minimum à une exception bad_alloc. Au final, tout ce qui invoque un constructeur, une méthode ou un opérateur est à proscrire d'un destructeur sauf si celui-ci capture les exceptions potentielles et n'en laisse échapper aucune. Il convient d'être extrêmement attentif, car en C++ les invocations de constructeurs et d'opérateurs sont souvent implicites.

■ Frédéric **Mazué** - *fmazue@programmez.com*

JAVA

Le livre de Java, premier langage

*Difficulté : ****
Editeur : Eyrolles
Auteur : Anne Tasso
Prix : 29,90 €

Java est votre choix ! Bravo ! Il ne reste plus qu'à l'apprendre. Vous commencerez d'abord, à travers des exemples simples en Java, à maîtriser les notions communes à tous les langages : variables, types de données, boucles et instructions conditionnelles, etc. Vous franchirez un nouveau pas en découvrant par la pratique les concepts de la programmation orientée objet (classes, objets, héritage), puis le fonctionnement des librairies graphiques AWT et Swing (fenêtres, gestion de la souris, tracé de graphiques). Vous découvrirez enfin comment réaliser des applications Java dotées d'interfaces graphiques conviviales grâce au logiciel libre NetBeans. Chaque chapitre est accompagné de deux types de travaux pratiques : des exercices. Le corrigé est fourni sur le CD-Rom d'accompagnement.

JAVA

Aide-mémoire Java

*Difficulté : ****
Editeur : Dunod
Auteur : collectif
Prix : 14,90 €

Etudiant ou amateur, ce livre est pour vous. Il se base sur Java et passe en revue : du modèle objet à l'environnement de programmation, des processus aux entrées-sorties, des API aux exceptions, de la généricité au graphisme. Chaque notion est illustrée par un ou plusieurs exemples et cas pratiques. Aucune connaissance en Java n'est pré-requise. Une bonne mise en bouche.

JAVA

Java 7

*Difficulté : ****
Editeur : Pearson
Auteur : Robert Chevallier
Prix : 22 €

Java 7 est disponible depuis quelques semaines, il serait temps de s'y mettre sérieusement. L'auteur couvre la version standard. Très pédagogique, ce livre fournit un cadre pratique pour acquérir une bonne maîtrise du langage Java. Il étudie chacune des notions de programmation objet : les classes, les objets, l'envoi de messages et l'encapsulation ; l'héritage, la composition et l'association de classes ; la réalisation de structures d'objets et d'interfaces graphiques ; la généricité. L'ouvrage revient sur l'ensemble des améliorations, nouveautés de cette v7. Les exercices, qui occupent la moitié du livre, sont intégralement corrigés. Tous les programmes étudiés sont opérationnels et montrent comment résoudre les problèmes posés. La progression pédagogique, les nombreuses illustrations et les exercices riches et variés font de ce livre un outil d'apprentissage et de révision indispensable pour les étudiants et les professionnels.

DESIGN

Responsive web design

*Difficulté : ***
Editeur : Eyrolles
Auteur : Ethan Marcotte
Prix : 12 €

Le titre Responsive web design vous intrigue et vous aurez raison. Comment concevoir des sites agréables et capables d'anticiper et de répondre aux besoins de vos utilisateurs ? En explorant des techniques CSS et des principes généraux de design, comme les grilles fluides, les images flexibles et les media queries, Ethan Marcotte démontre qu'il est possible d'offrir une expérience utilisateur de qualité, quelle que soit la taille, la résolution ou l'orientation de l'écran qui affiche le site. Un ouvrage original que nous recommandons à tout développeur web et webmaster.

ANDROID

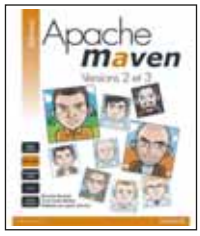
Développer des applications Android pour les nuls

*Difficulté : ****
Editeur : First
Auteur : collectif
Prix : 22,90 €

Android vous tente mais vous ne savez pas comment développer dessus ? Voilà peut être un livre qui va vous réconcilier avec la programmation et Android. Grâce à une approche claire et de nombreux exemples pratiques, vous apprendrez les fondamentaux de la programmation Android : les outils de développement, le SDK, l'architecture d'une application, la gestion des écrans, les animations, les données et leur gestion, les différents frameworks, le déploiement sur le marketplace... Bien entendu, si vous êtes déjà un développeur Java aguerri, vous irez vers un ouvrage plus technique mais celui-ci remplit très bien sa mission : faire découvrir la plateforme Android. Et c'est déjà bien !

VERSION

Maven versions 2 et 3



*Difficulté : ****
Editeur : Pearson
Auteur : collectif
Prix : 32 €

Maven est un des meilleurs outils open source pour le développeur. Et il se bonifie année après année. Maven est un outil de gestion et d'automatisation de développement Java qui a le vent en poupe. Les raisons : il systématise, rationalise et simplifie le développement collaboratif de projets Java, faisant gagner aux entreprises comme aux développeurs du temps et de l'argent ! Dans ce livre, les auteurs, tous spécialistes de Maven, explorent les concepts fondamentaux de Maven et leur mise en oeuvre pratique sur un projet, les fonctionnalités plus avancées pour les gros projets d'entreprise, les fonctions avancées, toutes les nouveautés de la v3. Incontournable !

DESIGN

Programmation concurrente en Java

*Difficulté : *****
Editeur : Pearson
Auteur : Brian Goetz
Prix : 24 €

La concurrence n'a jamais été un thème simple et Java ne facilite pas la chose. La programmation concurrente est indispensable dans un contexte parallèle et de multicore. Le développement, le test et le débogage d'applications multithread s'avèrent en effet très ardus car, évidemment, les problèmes de concurrence se manifestent de façon imprévisible. Ils apparaissent généralement au pire moment - en production, sous une lourde charge de travail. Une excellente approche pour mieux comprendre les mécanismes, la logique de la concurrence.

NetAdvantage®
ULTIMATE

RAPPORTS, VISUALISATION DES DONNÉES ET CONTRÔLES D'INTERFACE UTILISATEUR POUR ASP.NET, WINDOWS FORMS, JQUERY/HTML5, WPF,SILVERLIGHT ET WINDOWS PHONE 7

SCANNEZ ICI POUR DECOUVRIR ULTIMATE!
www.infragistics.com/ult

FAITES PASSER VOS APPLICATIONS AU NIVEAU SUPERIEUR
INFRAGISTICS.COM/ULTIMATE

INFRAGISTICS

KNOW formation

Allier le savoir-faire technique
au savoir-être relationnel,
pour le management et les équipes IT

Formation

pour dirigeants et équipes IT

✔ *Gérer le* **TEMPS**

✔ **MANAGER** *les projets*

✔ **COMMUNIQUER**

www.know-formation.com

CONTACT :

Stéphanie Khalif-Vennat : stephanie@know-formation.com - Tél. 01 74 70 48 91 - Fax 01 41 39 00 22

***Know-Formation**- Tour Albert 1er - 65, avenue de Colmar - 92500 Rueil Malmaison*