PROGRAMMEZ !
Le magazine du développement
PROgrammez !
www.programmez.com
mensuel n°132 - juillet-août 2010

Spécial Smartphone

Quelle plate-forme choisir ?

- Windows Phone 7 : la revanche de Microsoft
- Android : maîtriser les fonctions vocales
- Nokia, Blackberry, Windows Phone, Android, iPhone : **L'embarras du choix !**

Webmaster

Découvrez toutes les nouveautés de

Adobe AIR 2.0

ADOBE AIR

Communauté

Comment est né Eléphpant, mascotte PHP ?

eclipse

Préparez-vous à la révolution

Eclipse e4

Votre première application

iPad

Java

Apache Pivot : un RIA très prometteur

Windows 7

Utiliser les API multitouch dans vos applications

M 04319 - 132 - F: 5,95 € - RD

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 132 - juillet/août 2010

sommaire

éditO

Du silice pour nos neurones

Il serait peut être temps que nos pauvres petits neurones prennent un peu de repos durant l'été. Il faut dire que depuis la rentrée 2009, ils n'ont guère eu le temps de se mettre sur pause. Mais c'est aussi un bon moment pour réfléchir, se pencher sur des technologies. Des discussions avec des confrères et acteurs du marché ont été l'occasion d'affiner des réflexions.

Aux Métaphores, tu te soumettras !

Une idée fait son petit chemin chez quelques éditeurs et communautés de langage, la « programmation par métaphore ». Un des exemples mis en avant est le langage Kodu, issu des laboratoires de Microsoft. Au lieu de passer par des boucles, fonctions, commandes, etc. on utilise des blocs prédéfinis illustrant une métaphore comme 'avance', 'bouge ceci', 'anime de gauche à droite', etc. Il y avait déjà eu le fameux HyperCard avec les métaphores de la pile et des cartes. La question est de savoir si la programmation par métaphore ne devrait pas être réservée à de nouveaux modèles comme la domotique ou la robotique. En robotique, ce type d'approche est tout à fait appréciable pour le grand public, on l'a vu avec le logiciel de programmation de Lego Mindstorm. Cette approche peut aussi plaire aux utilisateurs d'entreprise pour créer des rapports, de petites applications métiers, sans faire appel au département informatique. Par contre pour le développeur, l'approche nous laisse sceptique.

Aux barres parallèles tu t'essayeras.

A force de dire que la programmation parallèle arrive, est incontournable, que le multithread s'impose partout, il serait peut être temps de s'y mettre réellement. Mais là, il y a plusieurs problèmes qu'il faut résoudre pour que le développeur se sente réellement concerné : simplifier le modèle de développement, monter en abstraction, ne pas devoir utiliser le C++.

Il faut que les langages de haut niveau intègrent des fonctions parallèles pour simplifier au maximum le parallélisme du code, ou s'appuient sur des frameworks et librairies que le développeur voit à peine. Mais, l'illusion ne doit pas faire illusion. Cette abstraction de haut niveau pose un problème : comment optimiser, comment faire du tuning sur un code « abstracté » ? Et cela impose une rigueur d'architecture énorme : attention aux fuites mémoires, aux problèmes de synchronisation… Et pour notre part, une approche hybride (natif + haut niveau) serait une des solutions des plus intéressantes, en évitant d'aller trop vers le pur 'natif'. Mais il y a encore 12 à 18 mois devant nous avant de disposer des bons outils et librairies.

'Natif' tu feras ou universalité tu embrasseras

Enfin, une des constantes de la programmation, de la technologie, est de savoir s'il vaut mieux développer (et donc optimiser au mieux) pour une plateforme nativement, en exploitant les possibilités du système, du matériel ou s'il faut être le plus indépendant possible pour que notre projet fonctionne sur le plus grand nombre de plateformes.

On voit que ces deux logiques s'opposent radicalement sur le marché des mobiles et dès maintenant sur les tablettes tactiles. Impossible d'avoir un seul code source, un seul projet capable d'adresser Blackberry, iPhone, Android, Nokia, Windows Phone 7. Il faut à chaque fois réécrire, adapter, modifier le code, l'architecture. Si une approche universelle permet de fonctionner sur un maximum de terminaux, on se coupe des fonctions avancées. Et cette dualité pose question depuis 25 ans. Ce sera votre devoir de vacances.

■ François **Tonic**
Rédacteur en chef
ftonic@programmez.com

Le magazine du développement
PROgrammez!

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. Mazue, S. Sauré, D. Catuhe. **Experts :** D. Seguy, M. Chaize, Y. Yang, E. Vautherin, T. Lebrun, L. Labat, O. Penhoat, M. Hubert, S. Rhayour, S. Warin, J. Sautret, H. Douss, P. Cauchois.

Illustration couverture : © Apple, Blackberry, Sony Ericson, D.R.
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

Abonnement : Programmez 22, rue René Boulanger, 75472 Paris Cedex 10
Tél. : 01 55 56 70 55
abonnements.programmez @groupe-gli.com
Fax : 01 40 03 97 79 - 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

PROCHAIN NUMÉRO

N°133 septembre 2010
parution 1er septembre

✔ Agilité et productivité
Comment être plus efficace, en utilisant les bonnes idées des méthodes agiles, avec les bons outils ?

✔ Dreamweaver CS5
Découvrez les entrailles de l'outil phare des développeurs web.

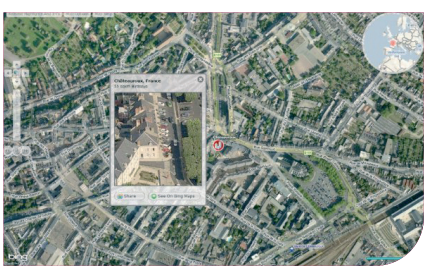
✔ Bases de données
Ce que nous préparent éditeurs et développeurs.

■ **Apple** a sorti iOS 4.0 le 21 juin dernier et le terminal iPhone 4 est disponible depuis le 24. Le système 4 apporte de nombreuses nouveautés sur les API, les performances, le multitâche, la personnalisation. Côté matériel, iPhone 4 inaugure une nouvelle gamme au design revu. On notera aussi une fonction vidéo conférence, des performances en hausse et une meilleure autonomie…

■ Le framework multiplateforme mobile, **PhoneGap**, dont iPhone qui permet de convertir en application native une application web devrait pouvoir continuer à exister sur iOS 4.0. C'est une bonne nouvelle et les développeurs pourront continuer à l'utiliser pour créer leurs applications ! Site : http://www.phonegap.com/

■ **WordPress**, un des plus célèbres moteurs de blogs et de gestion de contenu, est désormais en version 3. Cette dernière a demandé 6 mois de développement. Les extensions et thèmes WordPress 2.9 devraient continuer à être compatibles. La gestion du multisite a été améliorée ainsi que les gestions de contenu et la personnalisation. Site : http://www.wordpress-fr.net

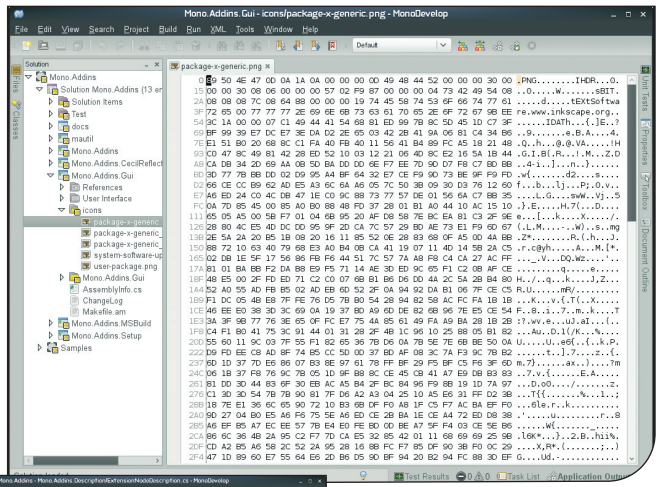
■ Le codec vidéo **VP8** fait désormais partie des formats supportés par le projet FFmpeg (depuis la v0.6). Cet outil est l'un des plus connus du monde open source. On pourra ainsi convertir des vidéos H.264 en WebM…

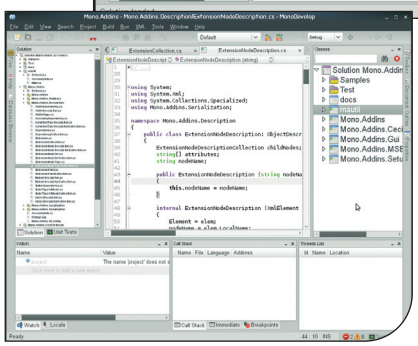


■ Un kit de développement pour **Bing Maps** sera disponible pour les développeurs fin juillet. Cette annonce est importante car Microsoft veut concurrencer Google Maps. Site : http://connect.microsoft.com/bingmapapps

IDE
MonoDevelop 2.4 : l'autre IDE .Net !

L'IDE phare pour développer avec la pile .net open source, Mono, vient d'être dévoilé en version 2.4 (mi-juin). Cette mouture contient de très nombreuses nouveautés. Il est disponible sur Windows, MacOS X et Linux et on dispose de ASP.Net, C# comme principaux langages .Net supportés. Cette version touche tous les aspects de l'outil : côté éditeur : nouvelle ergonomie, nouveaux assistants, meilleur support de MSBuild, éditeur de source plus élaboré, debugger plus complet sur l'affichage des valeurs, intégration d'un éditeur Hex ou encore d'un Template pour T4. Côté ASP.Net, on bénéficie de la complétion de code et du support des Web References.





Parmi les composants externes, notons les améliorations annoncées sur Vala, NUnit, MonoTouch et un important problème sur le support de MacOS X. Sur Vala, tout le moteur de complétion a été refait.

La partie mobilité connaît aussi quelques améliorations avec les nouveautés autour de MonoTouch : support de iPhone SDK 3.2 et 4.0, possibilité de faire des applications universelles et iPad, et une meilleure utilisation du iPhone Simulator Target. Actuellement le projet est en version 3.0.6. Parmi les nouveautés : support de iAD, Game Center. La 3.0.7 est annoncée lors de la sortie effective de iOS 4.

Le projet Mono travaille aussi à la finalisation de Mono Tools for Visual Studio 2.0. Il s'agit d'extensions payantes pour VS 2008 et 2010 pour pouvoir développer, builder et déployer des applications Mono. La v2 est annoncée comme une version majeure avec un nouvel outil de debug (celui de MonoTouch). Concernant le projet open source Silverlight, Moonlight, les développeurs travaillent à sortir la v3. Cette version sera compatible 32 et 64 bits sur Linux. Elle intègre désormais l'ensemble des fonctions prévues dans la version finale.

NetBeans 6.9 : disponible

Cette mouture très attendue apporte de nombreuses nouveautés. Nous notons le JavaFX Composer qui est un concepteur graphique pour les applications JavaFX GUI, similaire au Swing GUI Builder, l'interopérabilité OSGi, la plate-forme NetBeans et le support du développement de bundles OSGi avec Maven. NetBeans 6.9 c'est aussi le support du framework Spring 3.0, de GlassFish 3.0.1, du Framework Zend PHP, de Ruby on Rails 3.0, de C/C++ avec intégration des unités de test pour les projets C++. Bref, un cru très important pour la plateforme avec OSGi. La partie web est plus que jamais présente avec JavaFX, et les langages Web, dont Rails 3. Un peu « exotique », la pile Java Card qui se renforce. À l'heure où nous écrivons, nous n'avons pas de nouvelles de la prochaine version mais les premières préversions apparaîtront dès cet été !

Cependant, même si les développeurs semblent beaucoup apprécier NetBeans (en France, cela se remarque moins), il ne faut pas que le projet reste trop ancré dans l'actuelle plate-forme, car Eclipse avec le projet e4 va entièrement refondre son approche. À NetBeans de montrer qu'il est capable de se révolutionner sans doute avec la v8 mais pourquoi pas dès la v7…

LES CONTRÔLES LES PLUS RAPIDES...

Chez Infragistics, nous nous assurons que nos contrôles pour .Net vous permettent de créer les meilleures interfaces utilisateur possible. C'est pourquoi nous avons testé et retesté nos Data Grids afin de nous assurer qu'elles sont les plus rapides sur le marché et que nos Data Charts surpassent tout ce que vous avez expérimenté. Utilisez nos contrôles et vous obtiendrez non seulement le temps de téléchargement le plus rapide mais aussi des applications qui sont toujours attrayantes. Rapide et belle... C' est une « Killer app ». Essayez les vous-même sur **infragistics.com/wow**.

Infragistics
KILLER APPS. NO EXCUSES.

Infragistics Ventes France N° Vert 0800 667 307
Infragistics Europe Ventes +44 (0) 800 298 9055
twitter.com/infragistics

■ Les spécifications de **OpenCL** 1.1 ont été dévoilées. Parmi les nouveautés : mettre en queue des commandes depuis plusieurs threads, meilleure gestion de la mémoire, type vecteur à 3 composants, etc. OpenCL s'inscrit dans les développements parallèles afin d'améliorer la programmation des applications. Site : http://www.khronos.org/opencl/

■ La société **Tegesoft** vient de publier Camp en version 0.7. À la base, C++ est un langage qui ne propose aucune fonctionnalité d'introspection comme peuvent le faire des Python, Java ou C#, pour ne citer qu'eux. Tegesoft a eu l'idée de pallier ce manque en écrivant Camp. Au départ publié sous la licence GPL, Camp vient pour l'occasion de passer sous une licence plus permissive, la LGPL, qui autorise l'usage commercial.

■ **SQL Server 11,** nom de code Denali, commence à pointer son nez sur les documentations de Microsoft. Une version CTP devrait apparaître assez rapidement car plusieurs librairies font référence à une telle préversion. Aucun détail précis pour le moment sur Denali, sa date de sortie et les fonctionnalités.

■ **Ingres** a annoncé une version Linux de son projet VectoWise. Placée sous licence GPL, cette nouvelle technologie de base de données permet des analyses plus rapides, mais aussi plus fines et plus complexes. Pour ce faire, elle tire parti des avancées technologiques réalisées sur la partie matérielle (meilleure exploitation des processeurs de dernière génération et de la mémoire), mais aussi d'innovations telles que le rangement des données en colonnes (impactant sur la vitesse de lecture et les taux de compression), leur traitement vectoriel (utilisation simultanée de plusieurs moteurs de stockage), une meilleure exploitation du cache …

Concours

Adobe Challenge Air : sur un AIR de web

Durant 24 heures, Adobe France a rassemblé une quinzaine d'équipes dans un grand local à Paris. Le but ? Développer dans un temps limité une application utilisant les technologies Adobe ayant pour thème la famille.

Les équipes étaient des binômes, le plus souvent un développeur et un designer, parfois deux développeurs. Le défi ne fut pas négligeable pour plusieurs équipes qui durent changer de développement en cours de route, face à des problèmes techniques ou le manque de temps. Parfois le projet était trop ambitieux pour être réalisé en 24 heures laissant parfois de nombreuses fonctionnalités de côté. Parmi les équipes,aux côtés d'agences web, de freelances et de sociétés spécialisées dans le web, deux écoles étaient présentes : Epita (avec l'unique jeune femme engagée) et le DUT SRC de Bordeaux.
Chaque équipe avait sa propre vision des technologies et la manière de les utiliser. Globalement, tous les projets ont apporté des idées intéressantes, avec parfois, des concepts plus que séduisants : partage de photos avec intégration Flikr, logiciel pour gérer les listes pour la cuisine, les tâches quotidiennes, réseau social pour organiser des événements réels entre la famille et les amis ou encore pour aider l'étudiant dans sa nouvelle vie. Le choix du jury fut difficile. Nous avons activement soutenu l'équipe Freelance pour leur approche d'intégration avec Gmail, Maps, services web, etc. Ce fut notre coup de cœur. Mais nous avons aussi beaucoup apprécié l'approche de partage images et vidéos du projet Digitas avec un espace dédié, l'utilisation de token pour les sécurités ou encore la détection de volumes supplémentaires comme une clé USB et une bonne utilisation de AIR 2. Nous avons vu plusieurs projets ayant une approche tactile pour tenter l'innovation sur les interfaces comme le projet de SQLi. Bravo à toutes les équipes et à l'ambiance, malgré quelques ordinateurs « cramés » et les problèmes techniques. Côté design, le choix est allé à l'équipe belge pour Wip It, une gestion de tâche. Sans être révolutionnaire dans l'idée, l'application était fonctionnelle, terminée et complète, un atout pour le jury. Sur le prix technique, l'équipe Dagobert a astucieusement joué la carte du partage vidéo entre plusieurs personnes pour échanger des photos et des informations. C'est assez original et techniquement très bon (serveur Red5, Java, AIR 2).

LES GAGNANTS SONT :

1er prix : *Digitas*
Prix du design : *équipe belge*
Prix technique : *Dagobert*
Coup de cœur de Programmez : *Freelance*
Coup de cœur de la meilleure idée : *Epita*

Les projets sont visibles (avec les vidéos et présentations) sur : http://www.adobeairchallenge.com/

L'ambition d'Adobe France est de voir le challenge se développer ailleurs dans le monde. Nous avons été impressionnés par le niveau technique des équipes, notamment sur les idées d'interface ou l'utilisation de services externes pour se simplifier la vie et éviter de réinventer la roue. Place maintenant au vote du public !

agenda \\

JUILLET 2010

• Du 30 juin au 9 juillet, Sophia-Antipolis. **SophiaConf2010** pour découvrir, apprendre, échanger autour des derniers outils et techniques informatiques. http://www.sophia-conf2010.fr

• Les 08 et 9 juillet 2010,10,18 rue des Terres au Curé, 75013 Paris, **PHP Days "Industrialisez votre PHP!"**, pour monter en compétences sur PHP avec les experts français du PHP ! http://www.phpdays.com/

SEPTEMBRE

Du 09 au 10 septembre 2010, Paris – Porte de Versailles, Les pôles de compétitivité Cap Digital et Imaginove annoncent le premier événement européen sur la recherche et développement dans le domaine du jeu vidéo : **Future Game On**. http://www.futuregameon.com

• Le 22 septembre 2010, Séminaires Business Intelligence en Libre-service avec Microsoft SQL Server 2008 R2 et PowerPivot Microsoft France - Centre de Conférences 41 Quai du Président Roosevelt - 92130 Issy Les Moulineaux http://www.finelog.fr

• Du 30 septembre au 1er octobre 2010, Paris la 3e édition de l'**Open World Forum** rassemblera les décideurs mondiaux du numérique ouvert, sous le signe de l'innovation Libre et Open Source. 1 500 participants de 30 pays sont attendus pour cross- fertiliser les initiatives et dessiner le futur du numérique. http://www.openworldforum.org

Une vie de Geek

Eléphpant : 4 générations sous un même toit

Bonjour !

Je suis éléphpant, la mascotte PHP. Notez bien le P supplémentaire dans mon nom : certains me confondent encore avec mes cousins d'Asie, d'Afrique et des lave-autos, mais je suis bien distinct. Je suis difficile à prononcer, certes, mais unique en son genre.

Je suis né il y a quelques années, de la plume de Vincent Pontier. Grâce à son talent, il a transformé le P de PHP en ma tête et ma trompe, l'autre P en mon postérieur, et le H en mon ventre. Avec un peu de doigté, la finition a donné la fameuse silhouette d'éléphpant, que l'on croisait déjà autour de l'an 2000. Il a fallu encore quelques années avant que Damien Seguy harcèle suffisamment de PUG, les groupes d'utilisateurs PHP, et que soit lancée la première génération d'éléphpant. La grande famille des éléphpants compte maintenant plus de 8 000 membres, incluant 125 grands éléphpants. Cela représente tout de même presque une tonne de rembourrage. Peu d'entre-nous ont un véritable nom, mais quelques-uns parmi nous se distinguent :

- l'aïeul, qui est le seul à avoir les lettres PHP en bleu ciel. C'est le prototype de la famille, et le seul a avoir subi ce raté.
- 'Papa', qui est le premier gros éléphpant, et qui sert d'oreiller à Audrey : vous le reconnaîtrez facilement sur les conférences, il est un peu aplati. Papa commence à se faire vieux, et il sort moins de nos jours. Peut-être aurez-vous la chance de le croiser un jour.

D.R.

- Bien sûr, il y a 'Maman', le deuxième gros éléphpant de la famille, qui est chez Oriane. Ne me demandez pas comment on fait la différence entre les mâles et les femelles : demandez à un enfant, il saura vous le dire, à défaut de l'expliquer.
- Enfin, il y a les éléphpants 'rouges'. En fait, ils sont bleus, comme tous les autres. Ils portent ce drôle de nom à cause du logo Oracle, qui a

été appliqué sur le flan droit. Ils représentent la 2e génération.

Je suis l'enfant de la communauté PHP. Les générations sont organisées au fur et à mesure des besoins des groupes d'utilisateurs. Un appel est lancé pour rassembler des commandes, et produire d'un seul coup suffisamment d'éléphpants. Les groupes d'utilisateurs sont livrés par caisses de cinquante. Alter Way (et avant, Nexen Services) propose sa structure solide pour avancer les fonds de production, prêter les locaux de stockage et assurer la vente au détail. Elle se prête aussi à assurer la production pour d'autres sociétés, comme Oracle. Les groupes d'utilisateurs utilisent les hordes d'éléphpants pour attirer de nouveaux membres, ou se financer eux-mêmes. Le plus important est bien de combler les développeurs PHP, qui me reconnaissent spontanément.

Des développeurs posent nus avec moi

Les éléphpants permettent d'aller bien au-delà de la communauté habituelle des développeurs. Là où PHPunit parle aux développeurs, et où Drupal interpelle les webmestres, la petite peluche bleue fait parler de PHP là où il ne va pas d'habitude : il parle aux enfants, il attire les questions des femmes, et vous devriez voir la mine réjouit des voyageurs lors que Yannick. F. a transporté cinq d'entre nous durant les heures de pointe à Paris… Il y a plusieurs autres communautés libres et même des éditeurs, qui sont venus discuter avec nous pour savoir comment faire des peluches pour leur propre mascotte. Il paraît même que certaines grands-mères n'ont toujours pas compris ce qu'est PHP, mais sont très fières de me montrer à qui vient les voir.

Cela explique en partie que l'éléphpant soit l'espèce animale qui s'est répandue le plus vite sur Terre, plus vite même qu'Internet ! Avec la quatrième génération, nous sommes allés jusqu'en Antarctique ! Les cinq continents étaient déjà conquis grâce à la présence sur place de groupes d'utilisateurs : mes frères ont été envoyés dans des pays aussi étonnants que l'Angola, le Nicaragua, Aruba ou l'île Maurice. En novembre dernier, nous avons rencontré une biologiste marine, qui a accepté d'emmener l'un d'entre nous là-bas, et nous a ramené des photos exceptionnelles avec de vrais manchots. Il faut savoir que les militaires espagnols qui s'occupent de la base ont tellement aimé l'elephpant qu'ils ont demandé à le garder. Il nous reste maintenant à faire plus fort que l'Antarctique. Notre objectif actuel est d'avoir le premier elephpantonaute : le premier éléphpant à aller dans l'espace, et si possible, en revenir. Pour cela, nous recherchons activement des astronautes qui voudraient bien nous prendre avec eux. Si vous en connaissez ou avez des tuyaux, n'hésitez à me joindre ! Mais, sachez que le projet est déjà en route : des contacts ont été pris à l'Agence Spatiale Européenne. Nous avons plusieurs projets d'avenir pour notre famille : d'abord, au rythme de notre reproduction, nous serons plus d'éléphpants que d'éléphants en 2016! Ensuite, il y a une génération 5 qui est en cours de gestation, et que vous découvrirez à la naissance : mes papas souhaitent garder le secret jusqu'à l'heureux événement. Il y a toujours une génération d'éléphpants roses, destinés à celles qui développent en PHP. Elles sont encore trop peu nombreuses pour que cette génération voie le jour, mais nous nous approchons doucement du moment où cela se fera.

Bientôt à la télé

Nous avons d'autres projets un peu fous : notamment, nous voulons faire passer un éléphpant à la télé ou dans un film. En tant qu'accessoire visuel, il y a plusieurs séries comme *CSI Miami*, ou *the big bang theory* ou *IT crowds,* qui pourraient faire figurer un éléphpant sans dénaturer le show. Le but ultime est de faire passer l'éléphpant dans une production cinématographique, qui passerait dans une vraie salle. Si vous avez des idées, des contacts, nous aurons sûrement de quoi vous remercier.

Car la vraie nature de l'éléphpant est d'aller chercher le fun au coeur même de PHP. Adopter un éléphpant, c'est montrer qu'on est plus qu'un développeur, même patenté, passionné ou certifié : on fait partie de la grande famille de PHP, et ça nous plaît. Merci à tous les heureux phparents qui nous ont adoptés !

■ *Inspiré par* Damien **Séguy**

Votre site Web 1&1 tout com

AU MEILLE

Toutes les applications pour votre site Web

Avec 1&1, vous n'avez pas besoin de souscrire à une multitude de services. Tout ce qu'il vous faut pour développer vos projets sur le Web est inclus dans nos packs. Notre large sélection d'applications professionnelles vous laisse un maximum de liberté pour réaliser un site à votre guise et bâtir votre succès en ligne.

pris

3 MOIS GRATUITS*
SUR NOS PRODUITS DE RÉFÉRENCE

UR PRIX !

HÉBERGEMENT

1&1 PACK PRO STANDARD

- 3 noms de domaine INCLUS
- 10 Go d'espace disque
- Trafic ILLIMITÉ
- 1200 comptes email (POP3 et IMAP)
- 20 bases de données MySQL (100 Mo)
- Outils de création de site
- 1&1 Référencement
- 1&1 WebStat
- NOUVEAU : PHP 6 (bêta) et Zend Framework
- 1&1 Newsletter

~~9,99 €~~ **0 €** pendant les 3 premiers mois, après 9,99 € HT/mois (11,95 € TTC/mois)*

E-COMMERCE

1&1 E-BOUTIQUE M

- 1&1 e-Boutique Designer
- Trafic ILLIMITÉ
- Intégration eBay incluse
- Vos produits sur Kelkoo et Shopping.com
- 1&1 Référencement
- Paiement sécurisé via PayPal
- Jusqu'à 2000 articles et 200 catégories
- Module de gestion des commandes
- NOUVEAU : 1&1 Tableau de bord marketing
- NOUVEAU : 1&1 Codes promo

~~19,99 €~~ **0 €** pendant les 3 premiers mois, après 19,99 € HT/mois (23,91 € TTC/mois)*

Votre domaine à prix sensationnel :
le .fr à 3,99 € HT/an (4,77 € TTC/an),
le .info à 0,99 € HT/an (1,18 € TTC/an)* !

Consultez toutes nos offres du moment sur 1and1.fr !

* Offre « 3 mois gratuits » soumise à un engagement de 12 mois et à des frais de mise en service de 9,99 € HT (11,95 € TTC). Offres « domaines » valables la première année au lieu de 6,99 € HT/an (8,36 € TTC/an). Ensuite, les tarifs habituels en vigueur seront appliqués. Offres sans engagement également disponibles. Conditions détaillées sur 1and1.fr.

1&1

www.1and1.fr

Adobe AIR 2.0 : des applications Flash sur le bureau et sur mobiles

AIR, pour 'Adobe Integrated Runtime' est un environnement d'exécution multiplateforme (Windows, Mac, Linux), lancé par Adobe en 2008. Grâce à AIR, un développeur peut exécuter une application web en dehors du navigateur, directement sur le bureau des utilisateurs. Le fait de sortir de la sandbox du navigateur offre plus de liberté à une application AIR. Elle peut être lancée même si l'utilisateur n'est pas connecté à internet, elle peut accéder à des ressources en local (comme les fichiers du disque dur) et dispose de mécanismes réseaux plus avancés.

L'environnement AIR 2.0 embarque le Flash Player 10.1, le moteur Webkit pour rendre du HTML et une base de données SQLite (projet open source réputé pour sa légèreté et ses performances). Un développement Flash, Flex ou AJAX peut donc s'exécuter dans AIR grâce à ses différentes *stacks*. Le runtime connaît un succès grandissant sur le web mais aussi en entreprise. En effet, associée à des technologies Java comme LiveCycle Data Services, une application AIR peut détecter si l'utilisateur est online ou offline, et du coup synchroniser automatiquement les modifications effectuées sur les données en local avec le serveur. SalesForce.com, éditeur SaaS leader dans le CRM, utilise cette technologie pour les utilisateurs nomades de leur solution. Ainsi un commercial qui voyage en avion peut continuer de saisir de l'information dans le CRM, et celle-ci est automatiquement synchronisée avec le serveur dès qu'il se reconnecte. Adobe vient juste d'annoncer le lancement de la version 2.0 du runtime AIR et son lot de nouveautés. L'éditeur en profite pour annoncer l'arrivée du runtime sur les environnements mobiles et conserve ainsi la métaphore sur les PC. On parlera donc du Flash Player pour les applications qui s'exécutent dans le navigateur du mobile, et d'AIR si c'est une application native (disponible sur l'Android Market Place par exemple). Cet article va détailler les nouveautés de la version 2.0 et décrire les nouvelles API desktop et mobile.

De nouvelles API pour le mobile

Le fait d'annoncer le runtime sur Smartphones condamne l'environnement AIR à faire évoluer ses API et c'est chose faite avec la version 2.0. AIR embarque désormais le nouveau Flash Player 10.1, il dispose de nouvelles API essentielles sur mobile comme le Multi-Touch, la gestion de l'accéléromètre ou la géolocalisation. Le Flash Player 10.1 et le runtime AIR 2.0 résultent de plus de deux ans d'ingénierie pour optimiser l'exécution de Flash sur environnement mobile. Vous constaterez avec cette nouvelle version une nette amélioration du comportement de Flash en matière de consommation CPU et de gestion de la mémoire. Pour ce qui est des API, voici quelques exemples :

AIR 2.0 et le Multi-Touch

Pour créer une application Flex Multi-touch, et ensuite l'exécuter dans AIR, il faut utiliser la nouvelle API du Flash Player 10.1 qui peut gérer deux types d'entrées : la reconnaissance de gestes (GESTURE) ou la prise en compte des doigts en donnée brute (TOUCH). Le mode GESTURE permet de détecter automatiquement qu'un utilisateur effectue un mouvement de zoom (pinch), de pan, de swipe ou de rotation sur un composant. Le mode TOUCH permet de récupérer en donnée brute tous les doigts posés sur l'interface. Pour chaque doigt, Flash assigne un identifiant unique et vous renvoie la position du doigt, la pression sur l'écran et la surface du doigt. Le nombre de doigts réceptionnés par le Flash Player dépend directement de l'appareil utilisé. Un écran HP classique envoie deux doigts, un mobile renvoie normalement jusqu'à 5 doigts, et certains écrans 3M renvoient jusqu'à 20 doigts (pour utiliser vos doigts de pied ?). Si dans votre projet vous avez une image avec l'identifiant « monimage », voici comment surveiller que l'utilisateur effectue un mouvement de Rotation ou de translation (Pan) sur l'image :

```
package com.michael.prog
{
  import flash.events.TransformGestureEvent;
  import flash.ui.Multitouch;
  import flash.ui.MultitouchInputMode;

  import mx.controls.Image;

  public class monImage extends Image
  {
    public function monImage()
    {
      super();
      Multitouch.inputMode = MultitouchInputMode.GESTURE;

      this.addEventListener(TransformGestureEvent.GESTURE_ROTATE,
onRotate);
      this.addEventListener(TransformGestureEvent.GESTURE_PAN,
onPan);
    }

    private function onRotate(evt:TransformGestureEvent):void{
      this.rotation += evt.rotation;
    }
```

```
    private function onPan(evt:TransformGestureEvent):void{
      this.x += evt.offsetX;
      this.y += evt.offsetY;
    }
  }
}
```

Dans le constructeur, je déclare mon intention de travailler en Multitouch avec reconnaissance de gestes (le inputMode est GESTURE). Ensuite, je me mets à l'écoute d'événements de type « Geste de rotation » et « Geste de translation » sur mon objet. L'événement renvoyé possède des propriétés comme rotation ou offsetX, qui renvoient respectivement le dernier angle de rotation et le delta en X par rapport à la dernière position de l'objet.

AIR 2.0 et l'accéléromètre du mobile

De plus en plus de Smartphones sont équipés d'accéléromètres, qui permettent de comprendre la position dans l'espace de l'appareil. De nombreux jeux, comme DoodleJump utilisent l'accéléromètre pour commander le personnage principal. Si vous voulez recoder un jeu comme DoodleJump en AIR, vous devrez modifier la position du personnage principale en X, et tous les éléments du décor en Y. Penchons-nous sur le comportement du personnage principal qui dans mon jeu s'appelle Mylcare. Vous pouvez retrouver toutes les sources du projet sur mon blog (www.riagora.com) et découvrir comment recoder DoodleJump en 100 lignes de code ActionScript 3. Pour simuler le déplacement horizontal du personnage, il faut lui imputer une légère inertie. Voici la formule.

```
MyIcare.x -= (MyIcare.x - (MyIcare.x + accX * 10))*0.6;
```

Ici, les nombres 10 et 6 sont des coefficients que vous pourrez adapter à votre goût. Nous avons donc maintenant à gérer l'accélération en X, définie dans l'exemple par la variable accX. L'accélération est définie par l'inclinaison de l'appareil mobile, nous allons donc nous mettre à l'écoute de l'accéléromètre.

```
var myAcc:Accelerometer = new Accelerometer();
myAcc.addEventListener(AccelerometerEvent.UPDATE, onAccUpdate);

function onAccUpdate(evt:AccelerometerEvent):void{
        accX = evt.accelerationX;
}
```

Il suffit de déclarer un nouvel objet Accelerometer et de se mettre à l'écoute d'un changement (UPDATE). Lorsque le mobile change de position dans l'espace, la fonction onAccUpdate est appelée. Vous pourrez alors récupérer l'inclinaison sur les trois axes X, Y et Z. Dans le cas de notre jeu, seule l'inclinaison en X est intéressante. Pour débugger et tester vos applications Flash pour mobiles, sur votre PC, vous pourrez dans certains cas utiliser les émulateurs du constructeur. Sinon, je vous invite à tester Device Centrale CS5, qui est un environnement dédié aux tests d'applications Flash pour mobile. Vous trouverez la possibilité de simuler des actions Multitouch et de gérer l'accéléromètre à l'aide d'une représentation 3D de votre mobile.

AIR 2.0 et la géolocalisation sur mobile

Un autre plus d'une application mobile est la possibilité de récupérer la position de l'utilisateur dans le monde. La nouvelle API de géolocalisation du Flash Player 10.1 rend très simple l'usage de cette nouveauté.

```
var locale:Geolocation = new Geolocation();
locale.setRequestedUpdateInterval(5000);
locale.addEventListener(GeolocationEvent.UPDATE,   onlocation
Handler);

function onlocationHandler(e:GeolocationEvent):void
{
lastLat = e.latitude;
 lastLon = e.longitude;
}
```

Un objet Geolocalisation est à l'écoute d'un changement de position du téléphone mobile. Vous pouvez définir la cadence d'usage de la géolocalisation. Dans l'exemple, Flash va demander la position de l'utilisateur toutes les 5 secondes. L'événement de type GeolocationEvent transporte des propriétés de latitude et longitude. Associée à un composant comme celui de Google Map, la fonction de géolocalisation est redoutable de simplicité.

AIR 2.0 et les processus natifs

Selon moi, c'est la nouveauté la plus importante de cette nouvelle version. Classiquement, une application AIR peut lire et écrire dans des fichiers en local, et juste récupérer des documents glissés-déposés depuis le bureau de l'utilisateur. De nombreux développeurs souhaitent pouvoir communiquer avec des exe, lancer des lignes de commandes, etc., pour étendre les capacités de leurs applications Flash. C'est aujourd'hui possible grâce à l'API NativeProcess. De nombreuses entreprises disposent de librairies C, C++ ou Java qui s'exécutent côté client, pour communiquer avec un appareil de signature et de lecture de carte à puce, par exemple. Avec AIR 2.0, les développeurs peuvent embarquer ces librairies dans leur projet et communiquer avec elles pour retranscrire les résultats dans une interface riche. Voici les étapes pour communiquer avec un processus natif : la première étape consiste à vérifier si l'API NativeProcess d'AIR 2 est bien supportée.

```
if(NativeProcess.isSupported)
{
 launchEchoTest();
}
else
{
 textReceived.text = «NativeProcess n'est pas supporté.»;
}
```

Une application AIR est multiplateforme, mais si vous utilisez un Native Process, il est obligatoirement lié à un OS (et compilé pour une plateforme). Il faut donc détecter l'OS qui exécute votre application AIR, et pointer vers le bon exécutable. Ici, les exécutables sont stockés dans un sous-répertoire NativeApps du répertoire d'installation de l'application. Voici le contenu de la méthode launchEchoTest() :

```
var file:File = File.applicationDirectory;
file = file.resolvePath(«NativeApps»);
if (Capabilities.os.toLowerCase().indexOf(«win») > -1)
```

```
{
 file = file.resolvePath(«Windows/bin/echoTestWin.exe»);
}
else if (Capabilities.os.toLowerCase().indexOf(«mac») > -1)
{
 file = file.resolvePath(«Mac/bin/echoTestMac»);
}
```

La classe Capabilities permet de détecter l'OS hôte. La variable file pointe du coup vers le fichier exécutable. Il faut référencer en NativeProcess en se servant de la référence stockée dans file.

```
var nativeProcessStartupInfo:NativeProcessStartupInfo = new
 NativeProcessStartupInfo();
nativeProcessStartupInfo.executable = file;
process = new NativeProcess();
```

Le Native Process « process » peut se mettre à l'écoute de la sortie standard de l'exécutable (STDOUT).

```
process.addEventListener(ProgressEvent.STANDARD_OUTPUT_DATA,
 onOutputData);
```

Il faut aussi se mettre à l'écoute de l'entrée standard. Le processus natif émet un événement de ce type quand la donnée écrite en STDIN est flushée.

```
process.addEventListener(ProgressEvent.STANDARD_INPUT_PROGRESS,
inputProgressListener);
```

Tout est prêt, l'application AIR est à l'écoute, on peut lancer le processus avec la méthode start :

```
process.start(nativeProcessStartupInfo);
```

Pour que l'application écrive dans l'entrée standard :

```
process.standardInput.writeUTF(textToSend.text + «\n»);
process.closeInput();
```

Si le processus natif envoie des données sur sa sortie standard, l'application AIR peut les récupérer et afficher le résultat. Dans notre code, c'est la fonction onOutputData qui sera appelée. Voici comment récupérer ce qui est renvoyé par le processus :

```
textReceived.text = process.standardOutput.readUTFBytes(process.
standardOutput.bytesAvailable);
```

Voici un exemple de processus écrit en C, que vous pouvez embarquer dans votre application AIR. Ce programme va juste récupérer les données écrites en entrée standard, et les renvoyer en sortie standard pour engager un premier dialogue de courtoisie entre AIR et le processus natif.

```
#define BUFFER_SIZE 8192
int main(int argc, char** argv)
{
 char buf[BUFFER_SIZE];
 int  cnt;

 while ( !feof(stdin)
 )
 {
   cnt = read(STDIN_FILENO, buf, sizeof buf);
   if (-1 == cnt)
   {
     perror(«read»);
     exit(1);
   }
   if (0 == cnt)
   {
     // eof reached...
     exit(0);
   }
   write(STDOUT_FILENO, buf, cnt );
 }
 return 0;
}
```

AIR 2.0 et la détection de clé USB

Une application AIR peut aussi être à l'écoute d'un volume de stockage qui serait monté dans l'OS : une clé USB, un appareil photo, une caméra vidéo USB, etc. Un événement est levé et permet d'aller lire tout le contenu du volume. Vous pourrez ainsi très simplement lire les photos contenues dans une carte mémoire par exemple. Pour être à l'écoute d'un nouveau périphérique de stockage, utilisez les nouveaux événements StorageVolumeChangeEvent :

```
StorageVolumeInfo.storageVolumeInfo.addEventListener(Storage
VolumeChangeEvent.STORAGE_VOLUME_MOUNT, onVolumeMount);

StorageVolumeInfo.storageVolumeInfo.addEventListener(Storage
VolumeChangeEvent.STORAGE_VOLUME_UNMOUNT, onVolumeUnmount);
```

Quand un nouveau volume est monté, vous pouvez tester s'il agit bien d'un périphérique amovible et entamer le dialogue avec le volume.

```
private function onVolumeMount(e:StorageVolumeChangeEvent):void
     {
       if (e.storageVolume.isRemovable)
       {
         this.nativeWindow.activate();
         this.volumeName.text = e.storageVolume.name;
         this.targetVolume = e.storageVolume.rootDirectory;
         this.currentVolumeNativePath  =  this.targetVolume.
nativePath;
         this.start();
       }
     }
```

Si le volume de stockage est désactivé, la fonction onVolumeUnmount est appelée dans notre exemple.

```
private function onVolumeUnmount(e:StorageVolumeChangeEvent):void
     {
       if (this.currentVolumeNativePath == e.rootDirectory.
nativePath)
       {
         this.reset();
         this.mainView.selectedChild = instructionBox;
       }
     }
```

La variable currentVolumeNativePath permet de stocker sur quel

volume l'application travaille. Vous pouvez gérer ainsi plusieurs clés USB branchées pendant que l'application AIR tourne.

AIR 2.0 et le protocole UDP

Pour les développeurs avancés, sachez que vous pourrez aussi avec AIR2.0 engager des conversations UDP (Universal Datagram Protocol). Ce protocole est souvent utilisé pour une conversion chat vidéo, ou audio (VoIP), et les jeux multi-joueurs. Pour décrire le principe, nous allons développer un client UDP qui dit « Bonjour, comment ça va ? » à un serveur UDP qui répondra « ça flex merci ».
Le client va déclarer un datagramSocket et se mettre à l'écoute de données éventuelles sur le port 4455:

```
private var datagramSocket:DatagramSocket = new DatagramSocket();

protected function creationCompleteHandler(event:FlexEvent) : void {

 datagramSocket.bind(4455, «127.0.0.1»);

}

datagramSocket.addEventListener(DatagramSocketDataEvent.DATA,
dataHandler);
```

La fonction dataHandler affichera le contenu des messages transmis dans la console et fermera la conversation :

```
private function dataHandler(event:DatagramSocketDataEvent) : void {

 trace( event.data.readUTFBytes(event.data.length) );
 datagramSocket.close();

}
```

Pour envoyer un message au serveur sur un clic de bouton, exécutez ce code ActionScript 3 :

```
protected function clickHandler(event:MouseEvent) : void {

 var packet:ByteArray = new ByteArray();
 packet.writeUTFBytes(«Bonjour, comment ça va?»);

 datagramSocket.send(packet, 0, 0, «127.0.0.1», 4455);

 datagramSocket.receive() ;

}
```

Nous allons écrire un serveur UDP en Java qui communiquera donc directement avec l'application AIR. JAVA dispose aussi d'une classe DatagramSocket. Commençons par déclarer une simple classe :

```
public class SimpleUDPServer {
     public SimpleUDPServer(){ }

 public static void main(String[] args) throws IOException {

       DatagramSocket datgramSocket = new DatagramSocket(4455);

   }
}
```

Pour recevoir les données en JAVA, ce n'est pas comme dans AIR, il n'y a pas d'événement levé dès qu'un message arrive. Il faut donc se mettre à l'écoute en boucle des paquets UDP.

```
while(connected){
    try {

      byte[] buf = new byte[512];

      DatagramPacket incomingPacket = new DatagramPacket(buf, buf
.length);

      datgramSocket.receive(incomingPacket);

      }
  catch (IOException e) { }
```

Je vais ensuite tester la chaîne qui arrive pour répondre à AIR :

```
String s = new String(incomingPacket.getData(),0,incomingPacket.
getLength());

if(s.equals(«Bonjour, comment √ßa va?»)){

       String responseString = «√ßa flex merci»;

}
```

Le programme Java va ensuite détecter l'IP du client AIR et renvoyer le message dans un paquet UDP :

```
InetAddress address = incomingPacket.getAddress();
int port = incomingPacket.getPort();

DatagramPacket outgoingPacket = new DatagramPacket(buf, buf.
length, address, port);

datgramSocket.send(outgoingPacket);
```

Avec AIR 2.0, je vous rappelle que vous avez un accès direct au microphone de l'utilisateur. A vous de recoder un Skype Multiplateforme avec AIR ;)

AIR 2.5 et les smartphones

Adobe a annoncé l'arrivée de AIR sur mobiles, pour développer des applications natives Android, par exemple avec l'API Flash. Nous dévoilerons prochainement dans un article comment créer et packager des applications AIR pour Android. Pour l'instant, les informations ne sont accessibles qu'aux membres du programme bêta (sur labs.adobe.com). Adobe travaille avec Palm (HP), Blackberry, Nokia et Microsoft pour porter dès que possible la technologie sur ces autres plateformes. Vous pouvez aussi packager une application AIR pour iPhone grâce à l'iPhone packager, disponible gratuitement sur labs.adobe.com mais votre application ne sera certainement pas approuvée par l'Apple Store à cause des restrictions de l'éditeur. De nombreux développeurs peuvent utiliser le packager pour rapidement prototyper des applications sur iPhone, ou les utiliser pour des projets internes (sans passer par une validation Apple).

■ Michael **Chaize**

Eclipse E4 : le futur de la plateforme Eclipse est là !

Simple IDE à sa création, Eclipse est devenu au fil du temps une plateforme d'exécution complète, en sus d'être l'IDE de référence du monde Java. Du fait de sa popularité, un très grand nombre de développements ont été faits autour d'elle, que ce soit pour la création de plugins Eclipse ou bien pour des applications basées sur Eclipse RCP. Tous ces développements ont mis en avant un certain nombre de limitations dans la plateforme auxquelles la communauté Eclipse et les grands éditeurs la supportant ont choisi de répondre avec le projet E4, nom de code du prochain Eclipse 4.0. Tour d'horizon des grandes nouveautés apportées par E4.

Afin d'adresser les différentes problématiques remontées par les utilisateurs de la plateforme Eclipse 3.x, E4 a été pensé selon 2 axes prioritaires. Le premier d'entre eux concerne la facilité d'assemblage des différents composants de la plateforme Eclipse. En effet, un gros effort a été fait pour uniformiser les différentes pratiques dans ce domaine tout en offrant un rendu visuel plus attractif pour la plateforme Eclipse. Le second axe concerne la facilité de développement. Afin d'attirer de potentiels nouveaux contributeurs au projet Eclipse, il devenait primordial de rendre le développement de plugins Eclipse plus simple, mais également de l'uniformiser, puisque sur les versions 3.x il était bien souvent possible de réaliser la même chose de différentes manières sans qu'aucune documentation ne précise la meilleure pratique à employer. Ce manque de documentation est quelque chose de pénalisant pour les nouveaux arrivants sur la plateforme et nuit grandement à l'efficacité des développements afférents.

Modèle de programmation orientée Services

L'approche par composants provenant du modèle de programmation d'Eclipse 3.x demeure. Ainsi, le modèle de ce dernier reprend les grands principes de programmation d'Eclipse :

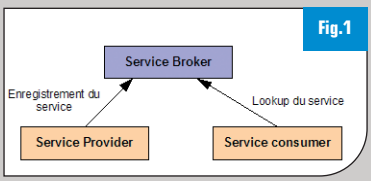
- Les Applications sont construites à partir de composants faiblement couplés, dénommés plugins ou bundles.
- Les Bundles peuvent être réalisés avec du code écrit en Java ou via d'autres langages. Ils peuvent être customisés, étendus ou liés à d'autres bundles via des points d'extension.
- La plateforme permet de définir un grand nombre de bundles mais garantit que seuls les bundles utilisés au runtime seront chargés et consommés par le système.

Au sein de ce modèle de programmation basé sur OSGI, on peut distinguer 3 participants : le service broker qui va créer un contexte permettant de gérer la collaboration entre le service provider et le service consumer. Dans le modèle 3.x (figure 1), le provider de ser-

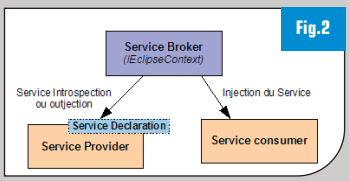
vice s'enregistrait auprès du service broker vers lequel le consumer de service devait effectuer un lookup pour récupérer un service. Ce modèle offre un découplage entre le service provider et le consumer mais nécessite d'avoir une bonne connaissance des services disponibles dans le broker de service.

Afin de tirer parti des avancées de ces dernières années en termes d'architecture logicielle, E4 se voit doté d'un modèle de programmation orienté Services (figure 2) au sein duquel le service broker peut désormais être vu comme un véritable contexte applicatif ayant la connaissance des différents services exposés par les providers et pouvant injecter ces derniers au sein des consumers qui les utilisent. Ce contexte permet notamment d'éviter de récupérer directement les fonctionnalités de la plateforme Eclipse via des singletons définis comme points d'entrées au sein des bundles. Pour cela, E4 utilise l'IOC en injectant les dépendances nécessaires directement au sein des classes. Le mécanisme d'injection de dépendances d'E4 est une implémentation compatible avec la JSR-330 de Sun. On retrouve de fait l'annotation @javax.inject.Inject qui permet l'injection de dépendances via des champs d'une classe, un constructeur et une méthode spécifique, mais également l'annotation @javax.inject.Named qui offre la possibilité de définir un *qualifieur spécifique* à un contexte d'objet. En outre, les annotations de gestion de services issues de la JSR-250 @PostConstruct et @PreDestroy sont supportées, ce que montre l'exemple suivant :

```
public class MyView {
	@Inject
	private IEclipseContext context ;
	@Inject
	private Logger logger;
	@Inject
	private ITimerService timerService;

	@Inject
	public MyView(Composite parent) {
	…
	}

	@PostConstruct
	private void init() {
		if (timerService != null) {
			// Do something
		}
	}

	@PreDestroy
	private void destroy()  {
		if (timerService != null) {
			// Do something
		}
	}
}
```



Modèle de programmation Eclipse 3.x



Modèle de programmation E4

Premiers pas avec E4

La première version officielle d'Eclipse E4 devrait sortir dans le courant du mois de Juillet. En attendant, vous pouvez d'ores et déjà tester la plupart des nouveautés qui en feront partie en récupérant la dernière milestone de E4 parue avant la release officielle. Il s'agit de la milestone 6 de la version 1.0 que vous pourrez récupérer à l'adresse suivante :

http://download.eclipse.org/e4/downloads/drops/S-1.0M6-201005232015/index.html

Le package se présente sous la forme d'une archive zip contenant tout ce qu'il vous faut pour utiliser E4, contrairement aux archives des versions 0.9 d'E4 qui nécessitaient un certain nombre de manipulations avant de pouvoir être utilisées. Afin de tester le projet Contact détaillé dans cet article, il vous faudra récupérer le projet ContactE4 sur le site de Programmez ! L'import se fait ensuite de manière classique via le Package Explorer d'E4. Une fois l'import fait, vous disposez d'un projet ContactE4 qui est un plugin Eclipse E4. Le lancement de ce projet se fait par la classe principale Application et vous permettra de mieux vous rendre compte des possibilités du framework XWT. En outre, pour tester le nouveau workbench amené par E4, il vous faudra créer un nouveau projet de type "e4 Application Project". Les différentes étapes de configuration du projet vous conduiront à définir le modèle de base de votre application ainsi que son fichier de mise en forme CSS. Le projet ainsi créé possède un fichier Application.e4xmi qui est un fichier au format XML définissant le modèle IHM de votre application. L'ouverture de celui-ci vous donnera accès à l'éditeur visuel d'Eclipse pour l'enrichissement du modèle sur lequel votre application s'appuie.

Le contexte du service broker est basé sur l'interface IEclipseContext et va bien plus loin que la simple injection de dépendances. Il offre la possibilité de gérer des contextes hiérarchisés au sein desquels d'autres services brokers sont supportés, ce qui lui confère une intégration parfaite avec OSGI et son service registry. De plus, les informations contenues au sein du contexte peuvent être enrichies à la volée. En ce qui concerne les providers de services, il est bon de préciser qu'ils doivent exposer leurs services en respectant le mécanisme de services d'OSGI. Ainsi basé sur ce standard de l'industrie logicielle, E4 s'intégrera parfaitement avec tous les frameworks publiant des services via OSGI, que ce soit au présent (Spring DM, iPOJO, …) ou à moyen terme, ce qui assure la pérennité de ce modèle. Enfin, dans un souci d'uniformisation des API et afin de limiter leur nombre, ce qui avait tendance à complexifier le développement sur la plateforme Eclipse 3.x, E4 vient avec une vingtaine de services applicatifs répondant à un grand nombre de besoins récurrents. Ceux-ci sont divers et variés puisqu'on va de l'API de Logging intégrée au service d'authentification en passant par un modèle de navigation ou des services de gestion de la persistance des états des IHM.

Modélisation du workbench E4

Le modèle IHM d'Eclipse 3.x, également appelé workbench, n'a cessé d'évoluer tout au long des différentes versions, 3.x devenant à chaque nouvelle mouture un peu plus flexible. Cependant, le résultat actuel est loin d'être satisfaisant et ce modèle basé sur un assemblage de composants se révèle complexe à l'usage et très difficile à maintenir au sein des applications Eclipse. Il faut également préciser

que les possibilités de customisation restent assez limitées et que par conséquent les applications basées sur Eclipse RCP ressemblent bien souvent à l'IDE original Eclipse. En effet, les principes de vues, perspectives, et éditeurs sont plutôt limitatifs voire complètement hors de propos dans certains contextes d'utilisation d'Eclipse RCP. Enfin, il existe bien souvent différentes manières de réaliser la même chose sans qu'aucune ne soit documentée comme étant la meilleure pratique à adopter. De fait, les applications Eclipse ne sont pas toujours très homogènes et le manque d'un modèle de base qui serait extensible se fait sentir. Pour pallier ce manque, le workbench E4 est modélisé par un modèle formel (figure 3) qui regroupe les éléments faisant partie d'une instance de workbench. Ce modèle reste à un certain niveau d'abstraction puisque le contenu des vues n'en fait pas partie, ce qui peut être l'un des seuls reproches qu'on pourrait lui faire [Fig.3].

Le modèle du workbench E4 a été écrit en utilisant Ecore qui est le langage de modélisation du framework EMF (Eclipse Modeling Framework). Standard de facto pour la modélisation sous Eclipse, EMF présente l'avantage d'être très léger au runtime (1.5 Mb) et extensible tout en étant largement optimisé, puisqu'utilisé par un très grand nombre d'éditeurs. Il est même une référence dans les démarches MDA/MDD en Java. En outre, EMF possède déjà tout l'outillage WYSIWIG nécessaire pour modéliser des applications Eclipse E4, ce qui permettra de nettement augmenter la productivité des designers d'applications basés sur la plateforme Eclipse. Enfin, l'utilisation d'EMF simplifie grandement l'écriture de code habituellement pénible, comme la persistance, l'écoute des événements de modification du modèle, …

Au sein de ce nouveau modèle extensible, la distinction entre vues et éditeurs n'existe plus et les perspectives sont optionnelles, ce qui vient lever une limite historique des anciens workbenchs décrits précédemment. Toujours dans un souci de simplicité d'utilisation, les Parts du modèle sont désormais de simples POJO Java possédant une URI afin de pouvoir y accéder au sein de la plateforme. Chose importante, ce nouveau modèle est disponible au runtime et les modifications qu'on lui apporte aussi bien en termes de structures qu'en termes de customisation sont appliquées à la volée.

Le dernier point majeur à retenir au sujet du workbench E4 concerne les renderers. Ces derniers vont permettre d'abstraire le workbench du framework IHM de rendu utilisé en transformant les éléments du modèle E4 en Widgets utilisables par la plateforme Eclipse. La granularité des renderers est assez fine puisque chaque élément du modèle a son renderer (spécifique ou standard) afin d'être rendu d'une manière adaptée en Widget. À charge par la suite au renderer de gérer le cycle de vie (création, binding model/Widget, rendu graphique, destruction) de l'élément graphique qu'il rend. L'implémentation par défaut des Renderers pour Eclipse E4 est basée sur SWT et elle peut être sélectionnée en tant que paramètre d'entrée au lancement.

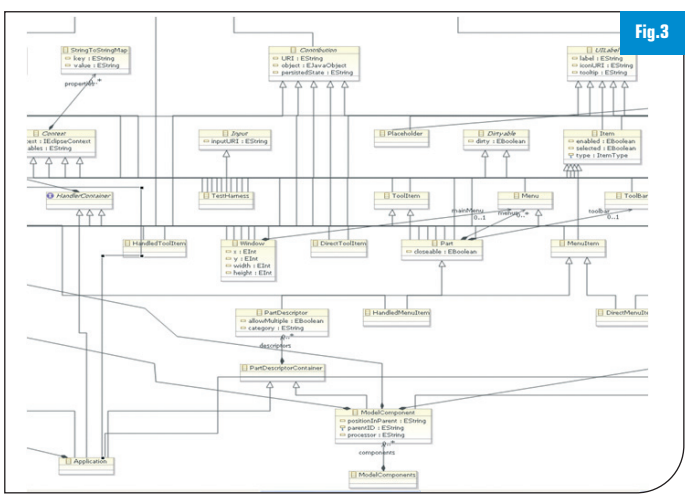
Mise en forme CSS

Afin de pouvoir réaliser des applications basées sur Eclipse RCP avec un rendu visuel plus approprié au contexte d'utilisation, la mise en forme aisée de composants Eclipse se révèle être un point crucial. Jusqu'à alors, la mise en forme des IHM se faisait soit en utilisant la très limitée API Presentation soit en dur, directement dans le code sur les Widgets. Ces solutions manquent cruellement de souplesse et surtout ne permettent pas une séparation entre la structure de l'IHM et son style comme on peut le voir sur les applications Web avec HTML et CSS.

Pour résoudre ce problème, E4 possède son propre moteur de style qui est utilisé pour customiser les différents Widgets produits par les différents renderers. Puisqu'il travaille sur des Widgets, ce moteur est indépendant du modèle IHM sous-jacent, ce qui lui permet potentiellement de pouvoir fonctionner avec des versions antérieures d'Eclipse. Le workflow de rendu visuel avec la mise en forme des Widgets est détaillé à la figure 4.

La mise en forme déclarative apportée par E4 est basée sur la syntaxe CSS qui est le standard Web. Un vaste éventail de propriétés CSS standard est supporté tel que les *fonts*, les *margins* ou les *colors* par exemple. En outre, E4 apporte des propriétés spécifiques et des pseudo-sélecteurs qui permettent de travailler directement avec les Widgets Eclipse. Les sélecteurs CSS de type sont basés sur le nom de classe du Widget concerné et les sélecteurs CSS via des class ou des ID sont également supportés. Enfin, les états sont pris en compte par le moteur de style en étant adaptés aux différents états du Widget que l'on sélectionne. Ainsi, l'exemple suivant permet de définir une couleur générale pour un Widget CTabFolder, mais une couleur différente pour un éditeur sélectionné en son sein :

```
CTabFolder {
      background-color : rgb(200, 200 ,200) ;
      font : normal ;
}
```



Modèle EMF du workbench E4

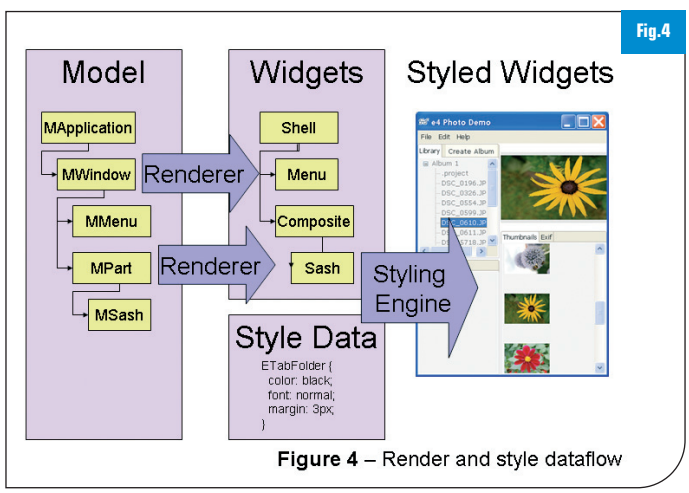


Figure 4 – Render and style dataflow

Processus de mise en forme dans E4

```
CTabFolder .editors :selected {
     background-color : rgb(255, 255, 255)  rgb(230, 230, 230) ;
}
```

Le workbench E4 n'est pas en reste et bénéficie également du moteur de CSS. Il est ainsi possible de le styler en lui précisant un fichier CSS externe que l'on souhaite lui associer en entrée. Les changements de style CSS sont appliqués au runtime et donc répercutés immédiatement sur les IHM. Enfin, il est également possible d'appliquer directement des styles sur les Widgets par programmation, bien que l'utilisation de fichiers CSS externes soit à privilégier.

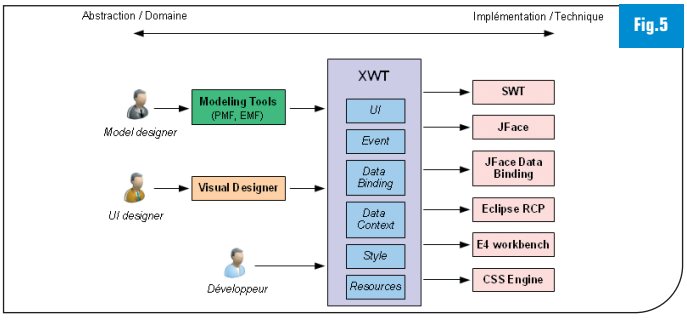
IHM déclaratives avec XWT

La facilité de développement prônée par E4 passe également par les IHM. En effet, la création d'IHM Eclipse via du code SWT s'avère bien souvent répétitive, diminuant d'autant la productivité des développeurs. En outre, le rendu visuel des Widgets est entremêlé avec le code de l'IHM, ce qui a pour effet de complexifier la maintenance de ces parties de code. Pour s'affranchir de ces différentes contraintes, le framework XWT a été intégré au projet E4.

À l'instar de ce que propose le langage MXML pour Flex, XWT est un langage XML permettant la définition de la structure hiérarchique complète d'une application ou bien d'un ensemble de Widgets. Un fichier Java, associé à la hiérarchie ainsi définie, va contenir la définition des méthodes callbacks ainsi que la logique métier. XWT se couple nativement avec le moteur CSS que nous avons évoqué précédemment. Le générateur d'IHM du framework se chargeant ensuite d'assembler toutes ces parties afin de créer les Widgets SWT/JFace correspondants. Ainsi, il bénéficie d'une séparation nette entre la structure d'une IHM (XWT), sa présentation (CSS) et la logique métier (Java) s'y rapportant.

Son architecture, détaillée à la figure 5, montre clairement que XWT va beaucoup plus loin en s'intégrant parfaitement avec le workbench E4 et le framework EMF d'Eclipse. Cette intégration réussie le positionne à la croisée de différents mondes : celui des modélisateurs métier, celui des designers d'IHM et enfin celui des développeurs. Afin de mieux nous rendre compte des possibilités de XWT que nous venons de détailler, nous allons prendre l'exemple d'un petit formulaire lié à un contact que nous allons modéliser via un simple bean Java. L'UI du formulaire est décrite simplement via le fichier XWT ContactUI.xwt présenté à la figure 6.

Il est intéressant de remarquer le binding sur les 2 champs texte avec les propriétés name et phone du bean Java associé au formulaire. Ce databinding, basé sur le framework JFace, est bidirectionnel. À charge ensuite au framework XWT d'associer le bean Contact passé dans le contexte de données au code SWT généré à partir du



Positionnement et Architecture de XWT

fichier XWT défini. Ce dernier est associé à la classe Java ContactUI qui contient la méthode callback submitContact, appelée lorsque l'on valide le formulaire via le bouton Submit :

```
public class ContactUI extends Composite {
  public ContactUI(Composite parent, int style) {
    super(parent, style);
  }

  public void submitContact(Event event) {
    // Récupération du contexte
    Contact c = (Contact) XWT.getDataContext(this);
    Button btn = (Button) event.widget;
    // Affichage de la popup montrant que le binding est bien réalisé
    MessageDialog.openInformation(XWT.findShell(btn),
            "Valeurs soumises", c.toString());
  }
}
```

Pour styler notre UI, nous définissions un fichier CSS externe minime nommé contact.css dont voici le contenu :

```
Button {
  font-weight : bold;
  font-size : 10;
  font-family : "Comic Sans MS";
}
Text {
  color : green;
}
Label {
  color : lightsteelblue;
  font-weight : bold;
}
```

Ce découpage met en évidence la séparation évoquée un peu plus en amont dans cet article entre la vue, le contrôleur associé à notre IHM et sa mise en forme. Enfin, l'assemblage de ces 3 composantes se fait au sein d'une classe principale avec un main de lancement :

```
public class Application {
  public static void main(String[] args) {
    // Récupération du fichier contactUI
    URL content = ContactUI.class.getResource("ContactUI.xwt");
    // Création d'un contact
    Contact contact = new Contact("Saurel", "0491010101");
    try {
      // Chargement du style via le CSS Engine
      CSSStyle cssStyle =  new CSSStyle(Application.class.get
Resource( "/sample/contact/css/contact.css"));
      // Ajout comme style par defaut dans XWT
      XWT.addDefaultStyle(cssStyle);
      // Lancement avec le contexte en entrée
      XWT.open(content, contact);
    } catch (Exception exc) {
      exc.printStackTrace();
    }
  }
}
```

Fig.6

```
<Composite xmlns="http://www.eclipse.org/xwt/presentation"
    xmlns:x="http://www.eclipse.org/xwt" xmlns:c="clr-namespace:sample.contact.ui"
    xmlns:j="clr-namespace:java.lang" x:Class="sample.contact.ui.ContactUI">
    <Composite.layout>
        <GridLayout numColumns="3" />
    </Composite.layout>
    <Label text="name" />
    <Text text="{Binding path=name}">
        <Text.layoutData>
            <GridData grabExcessHorizontalSpace="true"
                horizontalAlignment="GridData.FILL" widthHint="100" />
        </Text.layoutData>
    </Text>
    <Label></Label><Label text="phone" />
    <Text text="{Binding path=phone}">
        <Text.layoutData>
            <GridData grabExcessHorizontalSpace="true"
                horizontalAlignment="GridData.FILL" widthHint="100" />
        </Text.layoutData>
    </Text>
    <Label></Label><Label></Label><Button text="Submit"
        SelectionEvent="submitContact" bounds="54,41,44,23"></Button>
</Composite>
```

Fichier ContactUI.xwt

Le résultat de l'exécution de cette application est présenté à la figure 7. Il vient valider l'ensemble des concepts expliqués jusqu'ici concernant XWT et son intégration au sein d'E4.

Web To Desktop

Les dernières années ont vu un rapprochement très net entre les applications web et les applications desktop au niveau de la richesse des fonctionnalités proposées dans les IHM des applications de type web notamment. De ce fait, il devient de plus en plus souvent nécessaire de pouvoir disposer de composants logiciels fonctionnant dans les 2 mondes. Pour répondre à ce besoin, plusieurs possibilités ont été étudiées dans E4 et notamment celle de pouvoir réutiliser des composants logiciels écrits dans différents langages et surtout pour différentes cibles d'exécution. Plus encore qu'un rapprochement des mondes Web et Desktop, il est évident que la cible visée est un support multilangage au sein de la plateforme Eclipse.

Standard du Web de facto, Javascript a été l'implémentation privilégiée pour le support du multilangage dans cette première version officielle d'Eclipse E4. Le but de cette implémentation est de pouvoir tirer parti du meilleur du monde Eclipse dans Javascript et vice versa. Les faiblesses de Javascript sont assez nombreuses. On peut, par exemple, citer son manque de modularité ou d'extensibilité et une gestion des dépendances inexistante ainsi qu'un manque d'outillage criant. En Java et plus particulièrement sur la plateforme Eclipse, ces problématiques sont adressées via le système modulaire OSGI et son implémentation de référence Equinox. Pour appliquer le principe des bundles OSGI au monde Javascript, E4 apporte son propre framework Javascript implémenté en Java et considéré comme un bundle OSGI à part entière. Il est en charge du parsing des bundles Javascript et réalise la résolution des dépendances entre bundles. Les fichiers manifest de ces bundles sont au format JSON. En outre, comme le montre l'architecture s'intégrant autour du framework Javascript E4 (figure 8), le framework OSGI sous-jacent n'a aucune connaissance des bundles Javascript et de leurs

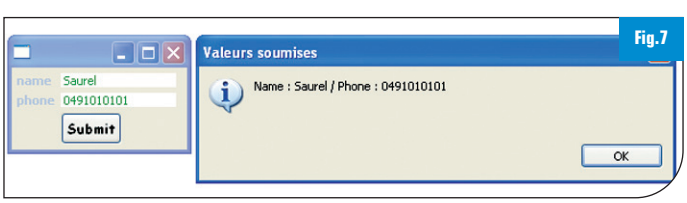


Fig.7

Exécution de l'application basée sur XWT

dépendances. L'installation d'un bundle Javascript dans le framework OSGI se fait soit par programmation, soit en utilisant le header spécifique "Javascript-Bundle". Il est important de préciser que le framework Javascript E4 peut être interrogé aussi bien par des bundles Java que par des bundles Javascript en passant à chaque fois par une API spécifique au langage appelant. Pour aller plus loin dans ce domaine, E4 est livré avec une réimplémentation complète du Plug-in Development Environment (PDE) en Javascript. Ce dernier peut être exécuté aussi bien dans un navigateur web que dans la plateforme Eclipse.

Divers

Au cours des réflexions Web To Desktop menées dans E4, il a également été exploré la possibilité d'un rapprochement Desktop To Web avec la mise au point de portages SWT vers des technologies orientées Web. Ainsi, la version 0.9 d'E4 propose un SWT Browser Edition permettant d'exécuter du code IHM SWT en tant qu'application Flash dans le navigateur. Ceci étant rendu possible par une cross-compilation du code SWT vers du code Action Script. Malheureusement, ce framework ne sera pas disponible dans la première version officielle d'Eclipse 4.0.

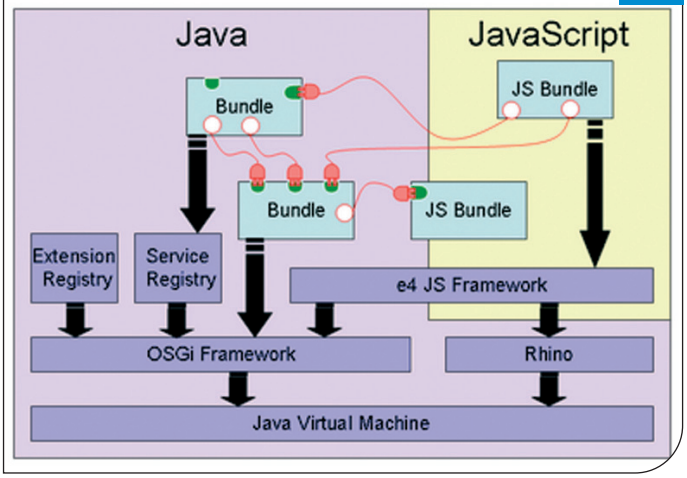
Dans le registre des nouveautés moins importantes incluses dans E4, on notera la présence du conteneur OpenSocial permettant d'intégrer au sein de la plateforme Eclipse, rapidement et simplement, des Gadgets Open Social. Enfin, et c'est une excellente nouvelle en vue des migrations futures vers E4, une couche de compatibilité a été développée et intégrée afin de pouvoir réutiliser les plugins Eclipse 3.x dans la nouvelle mouture.

Conclusion

L'effervescence créée au sein de la communauté par le projet E4 aura porté ses fruits. En effet, les importants efforts de développement réalisés par la communauté et les éditeurs auront permis à la fondation Eclipse de sortir dans les temps cette nouvelle version de la plateforme Eclipse. Très prometteuse et novatrice en bien des points, elle semble en mesure de gagner le pari fixé par ses fondateurs : être un socle solide et robuste pour les développements autour de la plateforme Eclipse pour les 10 prochaines années.

■ Sylvain **Saurel** – *Ingénieur d'Etudes Java / JEE*
ACP – www.acp-qualife.fr - *sylvain.saurel@gmail.com*

Fig.8



Architecture multi-langages E4

E4 et la IHM Déclarative

e4 est un projet d'incubation de la prochaine génération d'Eclipse. Il a pour but de tester et valider les différentes nouvelles technologies de la prochaine version.

Le développement IHM figure toujours parmi les tâches les plus coûteuses au sein d'un projet industriel. Qui plus est, la qualité d'IHM en termes d'aspect visuel et d'ergonomie reflète directement la qualité de l'ensemble du produit, ce qui attribue à l'IHM un rôle déterminant pour le succès du projet en question. Le développement IHM moderne ne s'arrête pas aux deux aspects de base : Création des Widgets et Traitement des événements. Il s'étend à d'autres aspects qui sont de plus en plus complexes :

- Gestions de liaison de données (Data Binding)
- Contrôle de sécurité
- Gestions de vues multiples, etc.

Les nouvelles techniques sont aussi de plus en plus utilisées dans les applications modernes telles que les effets de transition et les animations 2D et 3D. Le développement d'IHM nécessite de la connaissance et du savoir-faire autre que la seule programmation, comme la présentation graphique et la conception ergonomique.

Dans la version actuelle d'Eclipse (3.x), il existe un seul moyen pour réaliser les IHM : il s'agit de la programmation en Java par le biais d'un ensemble de bibliothèques d'IHM très riches, comme SWT, JFace, Data Binding, forms, draw 2D, GEF, Workbench etc. Mais la méthode de réalisation d'IHM en programmation présente désormais les inconvénients suivants :

- Exigence de connaissances techniques poussées sur les composants IHM
- Réutilisabilité faible
- Architecture incontrôlable
- Solution rigide et délais importants de mise en production
- Fonctionnalités des outils en développement visuel très limités
- Difficulté de connexion aux solutions de l'IDM (Ingénierie Dirigée par Modèle)

Un autre moyen de la réalisation d'IHM est devenu de plus en plus populaire depuis la naissance du Web. Il s'agit d'HTML, en fait l'ancêtre de l'IHM déclarative. Cette solution est largement utilisée par Mozilla dans son navigateur : Firefox. Une grande partie de l'IHM est décrite en XML, sous le nom de XUL.

La même solution a été mise en avant par le framework .Net de Microsoft à partir de la version 3.5 pour développer des applications de bureau. Puis depuis peu, elle est devenue la technologie de base de Silverlight pour la réalisation d'applications web riches. La spécification de ce langage déclaratif portant le nom de XAML est ouverte et publiée sous la Licence OSP (Open Specification Promise)

Concept d'IHM déclarative

L'IHM déclarative se distingue de la programmation "classique", par la flexibilité et l'abstraction naturelle. Les structures statiques d'IHM sont décrites dans un fichier externe souvent en XML, qui ne nécessite pas de compilation. De plus, cette architecture impose naturellement une séparation physique entre l'IHM et les traitements. Cette séparation est en fait une abstraction des concepts IHM [Fig.1].

Architecture

Si l'on analyse l'architecture des outils existants comme VE (Eclipse Visual Editor), on constate rapidement que la connexion aux composants IHM s'appuie directement sur le langage de programmation en Java. Cette architecture présente les problèmes suivants :

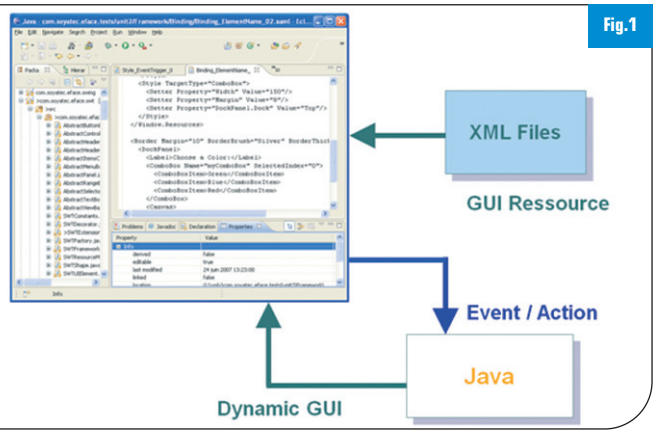
Description	Explication
Réalisation difficile et longue	Décodage des informations IHM à partir des classes Java Réécriture d'IHM en Java pour sauvegarder
Fonctionnalités contraignantes	Il est difficile de capter des informations complexes, par exemple, la liaison de données avec les mécanismes de validation et conversion de données.
Stabilité faible	Java class n'est pas un fichier structuré. La moindre erreur rend le fichier inutilisable
Partage d'informations entre les outils quasiment impossible	Les informations supplémentaires sont stockées en annotation. Il manque un standard pour que tous les outils s'alignent.

La solution XWT propose de fournir une couche très fine au-dessus de toutes les bibliothèques IHM existantes. Elle unifie, de façon homogène et cohérente, tous les concepts IHM en un seul langage déclaratif de type XML. Il s'agit, en réalité, d'une abstraction et d'une rationalisation des solutions existantes [Fig.2 et 3].

Cette séparation permet de répartir les développeurs en deux équipes : consommateurs et fournisseurs de composants IHM. Les fournisseurs sont des ingénieurs IHM qui ont une connaissance profonde de toutes les solutions existantes, ils développent des composants réutilisables destinés aux ingénieurs métiers qui sont en fait des consommateurs. Le consommateur n'est pas censé savoir comment les composants sont réalisés, il a seulement besoin de connaître ses fonctions logiques [Fig.4].

Le langage de description IHM en XML facilite le développement des outils évolués et extensibles avec une architecture simple comme l'éditeur WYSIWYG . Il s'agit de la solution idéale pour compléter les solutions IDM (Ingénierie Dirigée par Modèle) dans le but de générer l'IHM avec des fonctionnalités riches. Il simplifie la réalisation des moteurs de génération de code ou de transformation de modèle. Soyatec participe et mène un projet de cette nature dans Eclipse : PMF (Presentation Modeling Framework). Une démonstration l'a figuré lors d'EclipseCon 2010, pour montrer le processus et la capacité de modéliser et générer une application e4 avec XWT de bout en bout. Son moteur de génération de code a été réalisé en EGF (Eclipse Generation Factory).

Fig.1



Présentation

XWT fournit une infrastructure qui permet de réaliser des composants réutilisables. Il intègre les techniques innovantes et les concepts nouveaux dans un système simplifié afin de rendre possible son utilisation sans la connaissance détaillée de son implémentation. Il ne se limite pas uniquement à la manipulation des Widgets comme toutes les solutions IHM classiques, il implémente les solutions plus évoluées telles que la liaison de données métier. On appelle les composants IHM destinés à présenter les données « la Présentation ». La Présentation de XWT est une unité IHM pour gérer les interactivités d'échange entre les opérateurs et les données métier par le biais des technologies IHM standard. Une présentation est en général composée de 3 éléments :

- Ressource IHM en déclaratif (Réf : 5 Resource IHM)
- Contrôle d'événement en Java (Réf 6: Gestion d'événement)
- Contexte de données (Réf : 7 Connexion aux données)

Le contexte de données indique le type de données à présenter. Les expressions de connexion aux données dans les IHM, dépendent directement du type de contexte de données.

Une présentation est caractérisée par :

Description	Explication
Indépendance	Une présentation est comme un électron libre, elle a son propre gestionnaire de ressources incluant la liaison de données, l'unicité de nom, etc.
Réutilisabilité	Une présentation est complètement réutilisable n'importe où si et seulement si le type de connexion aux données est respecté.

Ressource IHM

XWT adopte le concept de mapping dynamique entre la structure XML et le modèle IHM. En d'autres termes, XML de XWT respecte en fait un sous-ensemble de la spécification XAMLiv, il n'a pas de schéma prédéfini. Chaque élément XML correspond à une classe IHM en Java. Un attribut XML correspond à une propriété du modèle IHM en Java. Cette correspondance est établie par un service de méta modèle de XWT lors de l'exécution. Par défaut, les modèles IHM sont issus de l'introspection des classes Java [Fig.5].

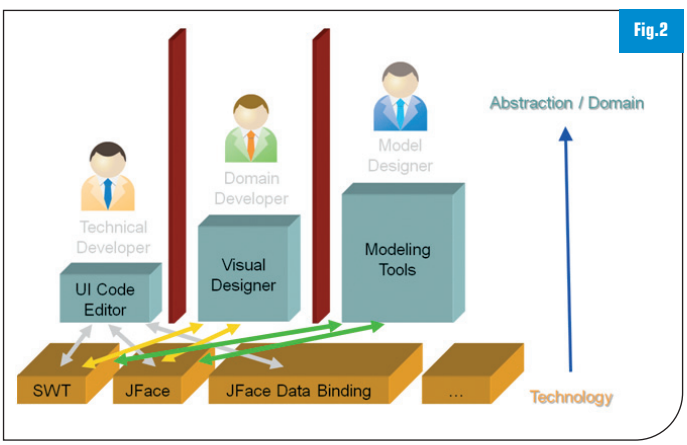


Fig.2

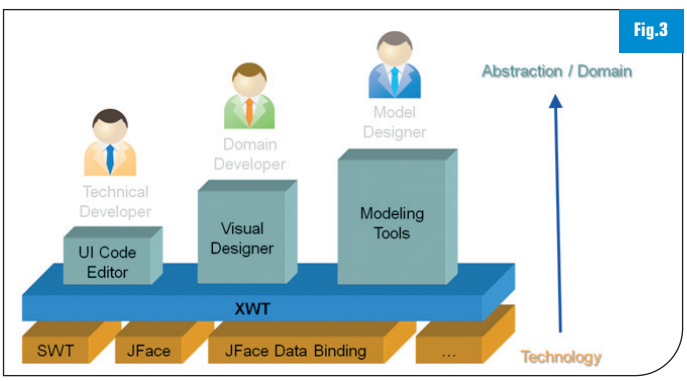


Fig.3

Note : *Il est important de noter que les fichiers XML sont utilisés uniquement pour la création des Widgets. Une fois ceux-ci créés, les informations déclaratives ne sont pas maintenues en mémoire, les développeurs peuvent alors récupérer les Widgets et appeler les API standard de SWT/Jface pour manipuler IHM. En aucun cas, XWT n'impose d'autres API d'IHM. Cela facilite l'adoption de cette technologie par les développeurs SWT/Jface, et aussi l'intégration et la migration des applications existantes.*

Par exemple, voici le code XML pour créer un bouton dans une fenêtre :

```
<Shell xmlns="http://www.eclipse.org/xwt/presentation"
    text="Appearance">
   <Shell.layout>
       <FillLayout/>
   </Shell.layout>
   <Button text="Hello world!"/>
</Shell>
```

Le défaut namespace est défini par « http://www.eclipse.org/xwt/presentation » qui représente la bibliothèque standard de XWT. La plupart des classes de SWT/JFace sont préenregistrées dans XWT. Leurs noms de classe peuvent être utilisés directement comme le nom d'élément en XML.

XML élément	Java classe
Shell	org.eclipse.swt.widgets.Shell
FillLayout	org.eclipse.swt.layout.FillLayout
Button	org.eclipse.swt.widgets.Button

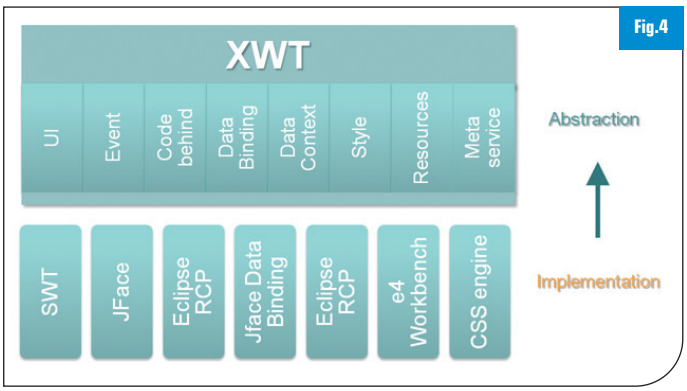


Fig.4

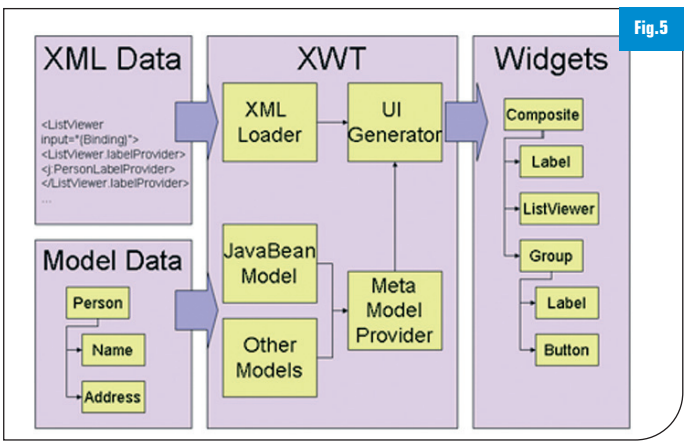


Fig.5

<Shell.layout/> est en fait un attribut « layout » de Shell exprimé sous forme d'élément XML. On peut charger ce fichier et créer les IHM via une méthode de la classe principale **XWT**

```
    URL url = …
    try {
        XWT.open(url);
} catch (Exception e) {
   …
}
```

Un fichier dont l'élément à la racine est de type Composte, peut être chargé dans n'importe quel Widget Composite :

```
    Composite parent = …
    URL url = …
    try {
        XWT.load(parent, url);
} catch (Exception e) {
   …
    }
```

En d'autres termes, grâce cette méthode, XWT peut s'intégrer dans les applications existantes développées en SWT/JFace.

Gestion d'événements

Un composant graphique de XWT est composé de deux fichiers : Resource Déclarative en XML et la classe associée en Java pour la gestion des événements.

L'exemple ci-dessous, illustre le concept de base par la création d'un bouton dans un Shell. Quand on clique sur le bouton, on exécute des traitements dans la classe Java.

XML
```
<Shell
  xmlns="http://www.eclipse.org/xwt/presentation"
  xmlns:x="http://www.eclipse.org/xwt"
  x:Class="ui.Handler">
  <Shell.layout>
   <FillLayout/>
  </Shell.layout>
  <Button selectionEvent="onClick"
       text="Click here"</>
</Shell>
```

Java
```
package ui;

import org.eclipse.swt.Event;

public class Handler {
  public void onClick(Object sender, Event event)
  {
    Button button = (Button) event.widget;
    button.setText("Hello");
  }
}
```

L'association entre ces deux fichiers est faite dans le fichier XML :
1. Définir un autre namespace, nommé en « x » par convention, dans un élément a pour but de gérer la connexion avec les langages de programmation http://www.eclipse.org/xwt.
2. Utiliser l'attribut x:Class pour indiquer la classe qui gère les événements
3. Déclarer le traitement d'événement dans Widget. Le nom d'attribut XML est composé par le nom d'événement SWT suffixé par « Event » pour se distinguer avec les propriétés de même nom. La valeur doit être le nom de méthode de la classe Java.

Liaison de données

La liaison de données est une technique nouvelle et très puissante pour gérer la synchronisation et l'interaction entre l'IHM et les données à présenter. XWT est réalisé en utilisant au maximum des bibliothèques standard d'Eclipse : JFace data binding. Pour plus d'informations, veuillez vous référer au site web du projet JFace data binding : http://wiki.eclipse.org/index.php/JFace_Data_Binding

La présentation de XWT unifie les concepts de connexion aux données aux composants IHM via une abstraction conceptuelle. Cette abstraction est matérialisée par un langage déclaratif, elle cache complètement l'implémentation et par conséquent, elle élimine la nécessité de la maîtrise de cette technologie pour les usages en masse. De plus, elle facilite de façon considérable la réalisation des outils tels que l'éditeur Visuel de type VE, le moteur de génération de codes.

Concept de base

Le concept de « contexte de données » (data context en Anglais) est le point d'entrée dans la liaison de données. Chaque élément IHM est susceptible d'avoir une donnée associée pour la présentation. Cette donnée peut être associée dans l'élément localement, ou le cas échéant, il peut se trouver dans son conteneur, directement ou indirectement suivant l'arborescence hiérarchique. La valeur locale surcharge celle de ses conteneurs. Pour cela, XWT a étendu le modèle standard de SWT en ajoutant une propriété « dataContext» dans la racine d'héritage : Widget. Dans l'exemple suivant, nous allons créer une présentation de Person qui a deux propriétés de type String: firstName et lastName. Ces deux propriétés doivent être présentées par deux Widgets Text respectivement :

Dans l'exemple suivant, on va créer une présentation de Person qui a deux propriétés de type String : firstName et lastName. Ces deux propriétés sont présentées par deux Widgets Text respectivement. Cette présentation est réalisée par le fichier nommé PersonView.xwt :

PersonView.xwt

```
<Composite
xmlns="http://www.eclipse.org/xwt/presentation">
  <Composite.layout>
      <GridLayout numColumns="2"/>
  </Composite.layout>
```

```
    <Label text="First name:"/>
    <Text text="{Binding path=firstName}"/>
    <Label text="Last name:"/>
    <Text text="{Binding Path=lastName}"/>
</Composite>
```

Cette présentation a un contexte de données implicite de type Person qui est associé au niveau racine. Ce contexte de données est partagé par tous les éléments. On constate dans les deux Text qu'il y a une expression spéciale dans le champ de saisie de Text : {Binding path=firstName}. C'est notre expression de connexion aux données. Cette expression indique que la valeur de propriété text de Widget Text est associée à la propriété firstName de contexte de données. Par défaut, cette connexion est bidirectionnelle : données vers IHM et IHM vers données. L'expression est délimitée par les accolades et commence avec le type d'expression Binding. Le path est une propriété de type Binding, où l'on spécifie le chemin de liaison par rapport au contexte de données. On peut aussi changer la direction de liaison en IHM vers les données en ajoutant une propriété mode :

```
{Binding path=firstName, mode=OneWayToSource}
```

Toutes les propriétés peuvent être la cible de liaison de données.

Conversion de données

Lors de l'affichage des données dans IHM, il se peut que les types soient différents. Supposons qu'on ajoute une propriété *age* de type *int* dans notre classe *Person*, pour afficher cette information dans un champ de saisie *Text* de type *String*, une conversion de données d'*int* en *String* est nécessaire. Heureusement, en standard, XWT dispose d'un ensemble de convertisseurs de données indispensables pour le chargement des fichiers IHM en XML. On peut étendre cette liste par les méthodes de XWT : XWT.addConvertor(), qui peut être utilisée systématiquement si besoin. On peut aussi spécifier un convertisseur local :

```
{Binding path=age, convertor=MyIntConvertor}
```

où MyIntConvertor est une classe Java qui implémente l'interface IDataConvertor.

Validation de données

Au moment de mettre une nouvelle valeur dans les données, il peut y avoir nécessité de valider pour s'assurer du respect de la contrainte d'intégrité. Dans notre exemple de *Person* avec la propriété *age*, la saisie des valeurs négatives doit être proscrite. Pour cela, on peut ajouter des contrôleurs de validation.

```
 {Binding path=age, validationRule=AgeValidationRule}
```

où AgeValidationRule est une classe Java qui implémente l'interface IDataConvertor.

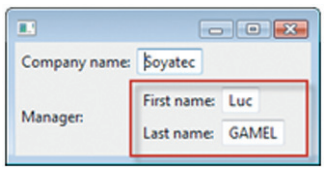
Traitement des erreurs

Lors de la synchronisation de données, en cas d'erreur, une notification est envoyée à l'utilisateur. Plusieurs méthodes s'imposent :

- Décoration
- Tooltip
- Boîte de dialogue
- Barre de statuts
- …

Le concept de traitement des erreurs de XWT est simple, étant aussi basé sur le concept de liaison de données.
Les états de liaisons sont maintenus dans un objet de type BindingContext. Un BindingContext défini dans un conteneur est partagé par toutes les liaisons de ses enfants. Il suffit ensuite d'établir la liaison entre ce BindingContext et IHM.

Réutilisation des Présentations



L'exemple suivant montre la réutilisabilité d'une Présentation dans une autre. Supposons qu'on dispose d'une classe *Company* pourvue d'une propriété *manager* de type *Person*. Dans la présentation de cette classe, pour afficher un contenu de manager identique à PersonView, il est préférable de la réutiliser directement.
Les composants IHM non standard peuvent être utilisés directement dans XML. Voici l'implémentation de cette présentation :

CompanyView.xwt

```
<Composite
  xmlns="http://www.eclipse.org/xwt/presentation"
  xmlns:x="http://www.eclipse.org/xwt"
  xmlns:j="clr-namespace:ui">
    <Composite.layout>
        <GridLayout numColumns="2"/>
    </Composite.layout>

    <Label text="Company name:"/>
    <Text text="{Binding path=name}"/>

    <Label text="Manager:"/>
    <j:PersonView dataContext="{Binding path=manager}"/>
</Composite>
```

- Déclaration d'un nouveau namespace qui représente le package en Java. La définition de namespace est composée de la manière suivante :

```
clr-namespace:<java package>
```

- Utilisation du nouveau namespace pour créer un élément de PersonView
- Basculement du contexte de données de Company à Person en utilisant une liaison sur la propriété dataContext.

C'est la méthode pour la réutilisation de tous les composants IHM non standard.

Conclusion

XWT est le fruit de 4 ans de recherche et développement de Soyatec. Cette deuxième génération de la solution IHM déclarative respecte une spécification standard et ouverte, et s'inspire d'une architecture mature de XAML définie par Microsoft. XWT est conçu dans des perspectives de rénovation des solutions IHM pour les besoins d'entreprise.

■ Yves **Yang** *(yves.yang@soyatec.com)*
Soyatec (http://www.soyatec.com), membre de fondation d'Eclipse, est une société française qui se spécialise dans l'IHM déclarative, la modélisation d'IHM et les outils d'IHM. Elle participe et mène plusieurs projets d'Eclipse : XWT, VEvi, ESL , PMFix, EGFxii etc.

Smartphone

Nous y sommes ! Depuis plusieurs années, Programmez ! traite dans ses pages de la programmation pour téléphone portable : WAP, J2ME, C++, etc., rien de bien excitant !
Mais depuis la sortie du premier Apple iPhone, c'est la révolution. La vraie !

La plate-forme à la pomme a secoué un marché jusqu'à présent atone, sans réelle innovation : terminaux lents et peu ergonomiques, une fragmentation du marché extrême, des modèles de développement de folie ! Souvenez-vous, pour concevoir des applications Java en J2ME, il fallait quasiment un binaire par famille de téléphone !

iPhone a été la tempête : plate-forme unifiée, modèle de développement unique, intégration matérielle + logicielle parfaite. Et très tôt, Apple a compris que le succès de son téléphone passerait par... les applications, donc par vous les développeurs ! Eh oui, le succès de l'iPhone n'est pas dû au matériel ou à la

attacks !

qualité de son système, même si les acheteurs y sont sensibles. Non, le succès vient des applications, de dizaines de milliers de logiciels disponibles dans une boutique en ligne, App Store. Et depuis, ce modèle logiciel, boutique en ligne, tout le monde le copie. Car c'est un modèle qui fonctionne.

Aujourd'hui, dans les téléphones dits intelligents (Smartphones), iPhone et Android dominent, même si le Blackberry reste devant. Microsoft va tenter d'enrayer sa chute avec le prochain Windows Phone 7 qui est une réussite, surtout si on le compare à la version 6.5... Microsoft joue son avenir mobile avec Phone 7. Et c'est pour cela, que l'éditeur encourage dès maintenant les développeurs à tester, à développer les applications et éprouver la Market Place pour les applications mobiles. N'oublions pas non plus Android. En pleine croissance, les téléphones Android se vendent très bien malgré un démarrage un peu poussif.
Nokia fait également de gros efforts pour promouvoir sa plateforme OVI.

Dans notre dossier d'été, nous avons voulu vous faire découvrir quelques fonctions peu connues comme les technologies vocales d'Android ou encore comment bien démarrer un projet Windows Phone 7. Mais surtout, nous vous proposons dès maintenant de coder pour une nouvelle génération de terminaux : l' iPad d'Apple. Aimé ou détesté, l' iPad va décupler la créativité du développeur !

Maintenant c'est à vous de jouer, de développer. Serez-vous iPhone, Android, Windows Phone 7, ou autre chose ? Yes, you can !

■ François Tonic

Nokia, Blackberry, Android, iPhone, Windows
Quelle plate-forme

Bon, maintenant que vous avez décidé de passer au développement sur mobile et plus précisément sur les téléphones intelligents (smartphones), encore faut-il choisir la plate-forme, et donc le modèle de développement. Que vous soyez développeur ponctuel à la maison, pour le fun, ou professionnel, plusieurs critères permettent de faire le ou les choix…

Aujourd'hui, difficile de cibler tous les Smartphones, pour une raison très simple : les modèles de développement, les outils, diffèrent d'un système à un autre ! Prenons le tableau ci-dessous de correspondance.

Il existe tout d'abord une sélection pour iPhone et iPad dans le sens où, il est nécessaire de développer les applications natives sur MacOS X avec XCode / Objective-C, même si pour le web mobile, la règle est plus souple… On attend de savoir si des environnements comme Titanium, MonoTouch ou Flash seront finalement autorisés ou non par Apple. Idem pour Windows Phone 7 qui nécessite avant tout un Windows équipé de Visual Studio 2010 et des outils Windows Phone. Même si rapidement, les environnements autres que Visual Studio seront sans doute disponibles.

Pour les autres, les principaux systèmes sont supportés par les IDE et SDK : Windows, Linux, MacOS X. Ce qui facilite la vie du développeur qui est habitué à son environnement de travail au quotidien ou qui ne veut ou ne peut investir dans une nouvelle plate-forme.

Le marché

Aujourd'hui, trois constructeurs se partagent le marché des Smartphones : Apple (iPhone), RIM (Blackberry), Google et les constructeurs (Android). L'ordre varie d'un pays à un autre. Ainsi aux Etats-Unis, selon Qantcast (mai 2010), les chiffres sont les suivants (part de marché selon la consommation web, Amérique du nord) : 58,8 % iPhone, 19,9 % Android, 10,4 % Blackberry, 10,9 % les autres… On peut constater une forte montée depuis un an des téléphones Android et une baisse de l'iPhone. Cette baisse est logique car Apple étant le seul constructeur à proposer l'iPhone et son système, alors qu' Android est disponible sur plusieurs dizaines de modèles ! La disponibilité de l'iPhone 4 devrait une nouvelle fois booster les ventes d'Apple, en attendant le nouveau Blackberry, orienté grand public, et surtout des gammes Windows Phone 7 ! Nokia souffre énormément de cette concurrence et a pris un grand retard sur le marché Smartphone mais le constructeur ne s'avoue nullement vaincu, tout comme Microsoft avec Windows Phone 7. Autre amer constat, la mort de la gamme Smartphone de Palm depuis son rachat par HP. Si on avait espéré une poursuite de la fabrication et des développements des Palm Pré et Pixi, HP a rapidement refroidi le marché. Le système de Palm, webOS, servira donc aux futures tablettes tactiles et peut être à des netbooks.

Bref, la guerre du téléphone intelligent ne fait que commencer. Surtout qu'Apple avec son iPad ouvre un nouveau front, celui des tablettes tactiles. Échec cuisant il y a quelques années, iPad peut relancer ce marché qui promet une mobilité toujours accrue à la maison, et en dehors, tout en proposant autre chose que le téléphone, et tout particulièrement sur la lecture de contenu, la vidéo, voire le jeu. Les usages sont à inventer, à confirmer.

Le choix de la plate-forme mobile va dépendre de vos affinités avec le langage, les outils, le ou les terminaux et le système mobile. Ainsi que l'effort d'apprentissage que vous souhaitez y mettre. Vous aurez toujours une mise à niveau à effectuer car l'interface change radicalement, et les API et les fonctions accessibles sont différentes d'un desktop Windows, Linux, MacOS X.

	Windows	Linux	MacOS X	Langage principal	soumission
Android	Oui	Oui	Oui	Java	Payant (ouverture)
Blackberry (1)	Oui	Non	Non	Java	payant
Windows Phone 7	Oui	Non	Non	.Net	Payant (annuel)
iPhone	Non	Non	Oui	Objective-C	Payant (annuel)
Nokia	Oui	Oui/non (2)	Oui/non (3)	Java C++	 Payant (ouverture)

(1) Uniquement pour la partie JDE, pour l'extension Visual Studio uniquement Windows
(2) dépendra du langage, du SDK, du système mobile cible
(3) dépendra du langage, du SDK, du système mobile cible

Phone 7 : choisir ?

Le langage et l'outil de développement

Blackberry

Editeur / constructeur : *RIM / RIM*
Système : *BlackBerry OS*
IDE principal : *JDE, Eclipse, Visual Studio*
Langage natif principal : *Java Micro Edition, C#*
Nb applications sur la boutique en ligne : *+ 7 000*

RIM, le constructeur des Blackberry, propose son propre environnement de développement pour Java Micro Edition : JDE. Actuellement en version 5, il supporte OpenGL ES 1.1, SQL Lite (pour les données), l'enregistrement de la vidéo et sa lecture. Voilà pour les principales nouveautés. RIM suit les évolutions de la plate-forme Java et notamment sur tout ce qui est interface, 2D et 3D, fonctions de transition entre les écrans, l'intégration des API et applications tiers. Bien entendu, il inclut un émulateur de terminaux. Sur ce point, il est à noter que le constructeur propose de nombreux profils pour les différents modèles. Au-delà de l'outil JDE, RIM propose aussi un plug-in complet pour Eclipse. On dispose également d'une approche web du développement. Pour cela, il s'appuie sur les standards du web et sur les IDE Eclipse et Visual Studio pour créer rapidement des widgets dédiés au Blackberry. JavaFX n'est pas supporté.

La partie widget est mise en avant par RIM. On peut accéder à de nombreuses fonctions du terminal tout en assurant une bonne sécurité. Ce n'est pas un hasard si le constructeur pousse les Super Apps. Elles permettent d'avoir une excellente expérience utilisateur, assurer une intégration native, être toujours connecté et intégré à un réseau social.

Aujourd'hui, RIM vise le grand public et d'ailleurs, l'entreprise représente moins de 50 % des ventes de terminaux. Preuve des ambitions du Blackberry pour contrer Android ou iPhone. D'autre part, RIM dispose d'une boutique en ligne pour les applications : App World. Bonne nouvelle, il n'est pas utile d'être développeur référencé au programme Blackberry Alliance pour soumettre des applications. Mais pour pouvoir soumettre une application, il faut s'acquitter de 200 $ US pour les frais d'administration et divers. Par contre si l'application n'est pas approuvée, l'argent vous est rendu. Cette somme est valable pour 10 soumissions d'applications. Et RIM examine les applications proposées et vérifie le respect des règles applicatives Blackberry.

Pour plus de détails sur la soumission : http://na.blackberry.com/eng/developers/appworld/faq.jsp

Sur la partie développeur : http://na.blackberry.com/eng/developers

iPhone

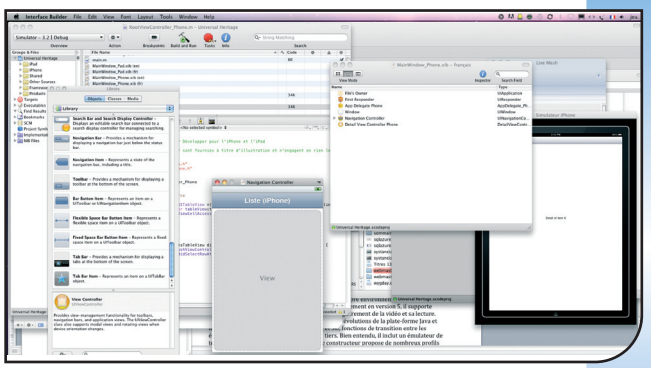
Editeur / constructeur : *Apple / Apple*
Système : *iOS (anciennement iPhone OS)*
IDE principal : *Xcode / Interface Builder*
Langage natif principal : *Objective-C*
Nb applications sur la boutique en ligne : *+ 200 000*

Pour développer des applications iPhone, un impératif (pour le développement natif) : avoir MacOS X ! Et s'inscrire au programme iPhone Developer Program pour un montant de 99 $ par an. Cela permet d'accéder aux dernières versions du SDK, de l'iOS, de l'ensemble des documentations (particulièrement nombreuses) et bien entendu de tous les outils de développement. Le développeur se servira de deux outils principalement : Xcode et Interface Builder. Une nouvelle version Xcode est en préparation qui devrait apporter beaucoup d'améliorations et une refonte totale de l'environnement avec peut- être la fusion Xcode – Interface Builder.

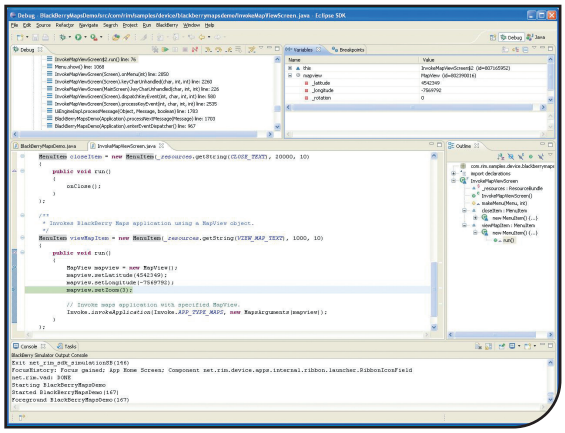
L'autre impératif pour développer une application native : Objective-C. Il s'agit du langage historique de MacOS X, hérité du rachat de NeXT. Pour le développeur habitué à C, C++, pas beaucoup de nouveautés, par contre pour un développeur Java ou C#, l'apprentissage sera peut-être un peu plus long car il faut assimiler la logique Objective-C et la structure des projets qui diffère de celle d'un projet Windows Phone ou même Eclipse.

Le processus de soumission est

La boutique Nokia



Environnement Xcode



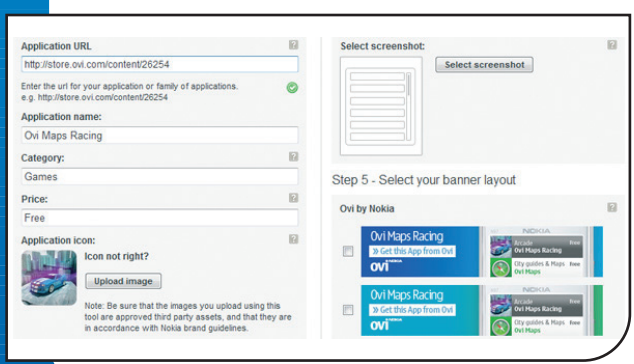
rigoureux notamment sur les questions liées au contrôle parental. Apple valide ensuite l'application, payante ou gratuite. Les refus sont motivés par des commentaires. Si le développeur les suit, il pourra soumettre de nouveau le logiciel.
Site : http://developer.apple.com/:

Nokia

Editeur / constructeur : *Nokia / Nokia*
Système : *Symbian, Maemo / MeeGo*
IDE principal : *Qt Creator*
Langage natif principal : *C++, Java, Python*
Nb applications sur la boutique en ligne : *?*

Nokia, un des plus gros constructeurs de téléphone, a multiplié les rachats : Qt, Novarra, Symbian. Aujourd'hui, Nokia propose plusieurs types de développements pour sa large gamme de mobiles : Python, Java et C++, sans oublier la partie VB et Flash.
L'une des difficultés avec Nokia est de déterminer la ou les familles de terminaux que vous souhaitez cibler car les outils, les SDK, les langages, les fonctionnalités ne seront pas les mêmes. Et le fait de disposer de plusieurs systèmes d'exploitation ne facilite pas le choix du développeur, hormis pour la partie web. Prenez votre temps pour regarder l'ensemble des opportunités.
Parmi le large choix d'outils et de SDK, Nokia propose :

- **Nokia Qt SDK** : il s'agit du framework bien connu multiforme Qt permettant de développer en C++ des interfaces très riches. On dispose bien entendu d'un simulateur et surtout de l'IDE maison, Qt Creator. Il permet de développer pour Symbian, Maemo et MeeGo. Pas de support de MacOS X.
- **WRT** : ensemble de plugins orientés



développement web pour IDE Aptana, Dreamweaver, Visual Studio. Il supporte HTML, CSS, JavaScript. On peut rapidement créer des applications web de type widget pour les téléphones Nokia. On dispose des accès aux API et fonctions Nokia (Platform Service 2, API en javascript et API Bridge pour faire le lien entre le matériel et WRT) ainsi que d'une librairie de type jQuery. Pour faciliter la vie du développeur, Nokia propose aussi des templates de projets.

- **Java** (S40) : un des langages phares pour Symbian. On dispose de l'ensemble des Api Java Micro Edition, des interfaces 2D, 3D, Open GL. Le package Java est disponible pour NetBeans et Eclipse mais uniquement sous Windows. Il Inclut tout naturellement l'IDE, le debug, les simulateurs nécessaires. On dispose de différents SDK pour Series 40 et S60. Nokia propose aussi un Tools for SVG. Site officiel : http://www.forum.nokia.com/Develop/Java/
- **Ovi Service** : le SDK Ovi est la prochaine étape des applications Nokia utilisant un développement web (html, css, javascript). S'interface avec le WRT Tools.

Aujourd'hui, Nokia pousse WRT et Qt, contrairement à Java.
Les templates proposent 3 niveaux : low-end, mid-range et High-end. Le niveau va dépendre des gammes de mobiles et des fonctionnalités. Le High-end inclut des composants web pour l'interface (navigation, grilles, boutons, éléments de data source), fichiers CSS et JavaScript et les fichiers graphiques pour Adobe Creative Suite (pas encore de CS5). Il supporte le WebKit Touch, le moteur Maemo. Très pratique pour tester et optimiser selon le système. Le Template est indispensable aux développeurs pour Nokia.
Site Template : http://www.forum.nokia.com/Develop/Web/Mobile_web_browsing/Web_templates/Templates_for_high-end_devices/
Nokia pense aussi au design et dispose d'une section design très complète avec les bonnes pratiques, le process de création, l'expérience utilisateur, les librairies de design (selon la famille de mobiles, le langage).
Pour la partie boutique en ligne des applications, Nokia propose OVI. Le constructeur invite tout d'abord les développeurs à comprendre et à respecter les règles du OVI Store et les éléments à intégrer au projet et lors de la soumission de l'application (http://www.forum.nokia.com/Distribute/Ovi_Store_guidelines.xhtml). Et de la même façon, selon la famille de mobile, OVI n'apportera pas le même niveau de fonctionnalités comme sur la signatu-

	S60 2nd Edition SDKs	S60 3rd Edition SDK	S60 3rd Edition, Feature Pack 1 SDK	S60 3rd Edition, Feature Pack 2 SDK	S60 5th Edition SDK
Qt			x	x	x
Extensions Plug-ins		x	x	x	x
Language Plug-ins				x	x
Nokia Energy Profiler External APIs			x	x	x
Sensor API Plug-in				x	Integrated
Multiple Drive Support (MDS) Plug-in				x	Integrated
Open C/C++ Plug-ins		x	x	x	Integrated
Nokia Eseries SDK Plug-ins		x	x	x	
Sensor API Plug-in for Nokia 5500 Sport		x			
OpenGL ES 1.1 Plug-in		x	Integrated	Integrated	Integrated
Ethernet Plug-in	x	Integrated	Integrated	Integrated	Integrated
SIP Plug-in	x	Integrated	Integrated	Integrated	Integrated

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.

Cas clients

Actu triée par secteur — *Avis d'Experts*

Actus — *Evénements* — *Newsletter* — *Vidéos*

www.solutions-logiciels.com

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

Envoyer par la poste à : Solutions Logiciels, service Diffusion, 22 rue rené Boulanger, 75472 PARIS - ou par fax : 01 55 56 70 20

1 an : 30€ *au lieu de 36€, prix au numéro (Tarif France métropolitaine) - Autres destinations : CEE et Suisse : 36€ - Algérie, Maroc, Tunisie : 36€ , Canada : 48€ - Dom : 45€ Tom : 60€*

6 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 : 6 rue Bezout, 75014 Paris

SL014

re des applications ou encore sur les formats de packages à fournir. Bonne idée : on dispose d'outils marketing pour promouvoir et mieux vendre son application. OVI représentait plus de 1,5 million de téléchargements par jour (tout type de contenu et application). Pour soumettre des applications, l'ouverture du compte est payante : 50 €.

Concernant les programmes développeurs, Nokia propose 3 grandes catégories :

- **Forum Nokia Launchpad** : accès aux informations techniques, aux pré-versions des API, SDK et systèmes, aux opportunités marketing. Disponibilité de réduction sur le matériel. Accès prioritaire à l'OVI Store. Il faut être membre du Forum Nokia. Payant. Environ 300 € par an.
- **Forum Nokia Pro** : accessible uniquement sur invitation. Il autorise l'accès aux informations techniques, aux informations marché, etc. Très orienté éditeurs, ISV. On dispose aussi d'un support technique, des matériels en prototype.
- **Forum Nokia for universités** : dédié aux universités, étudiants.
- **Community** : c'est la communauté des développeurs Nokia avec les blogs, wiki, forums.

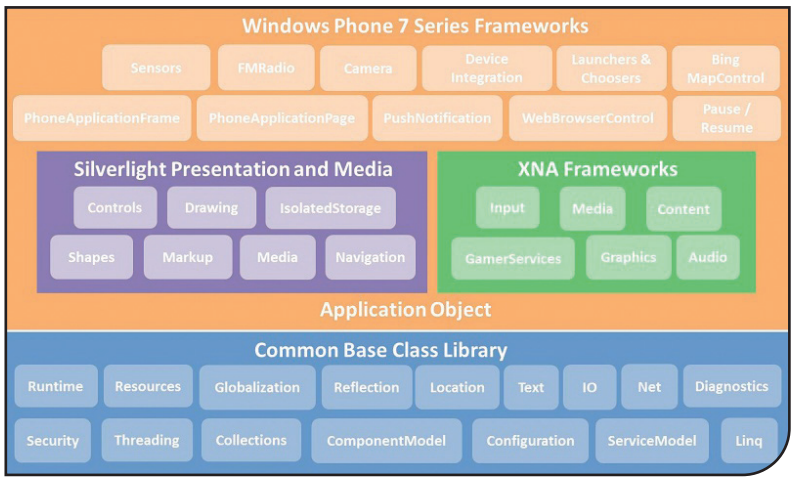
Le développeur peut s'inscrire (gratuitement) au Forum Nokia, la section développeur du constructeur. Site : https://www.forum.nokia.com/Profile/Join.xhtml

Windows Phone 7

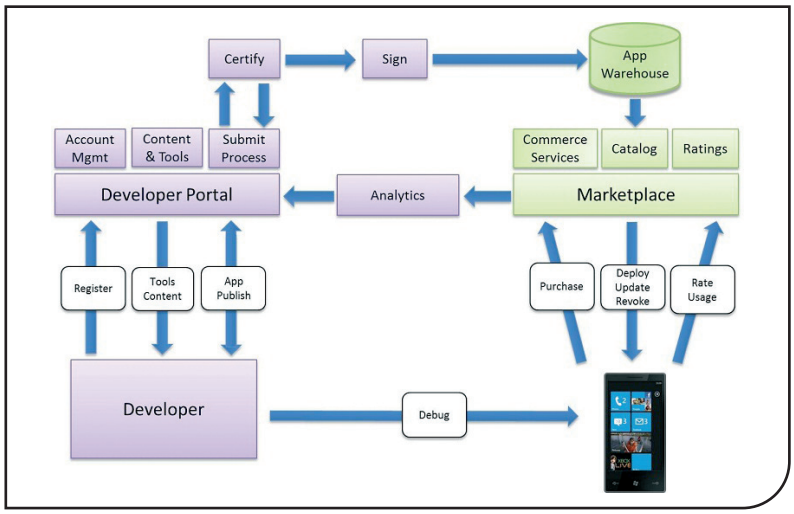
Éditeur / constructeur : *Microsoft / divers (pas disponible à la vente)*
Système : *Windows Phone*
IDE principal : *Visual Studio*
Langage natif principal : *.Net*
Nb applications sur la boutique en ligne : *?*

Dans quelques mois (automne 2010), Microsoft va sortir Windows Phone 7, successeur de la ligne 6.x. Pour l'éditeur, cette version est vitale pour garder une place importante sur le marché du téléphone mobile et tout particulièrement sur le Smartphone. Après une version 6.5 très mitigée, la version 7 promet un environnement entièrement refait, en partie réécrit, avec un nouveau modèle de développement, de nouveaux outils, une nouvelle boutique en ligne, du multitouch, une nouvelle interface utilisateur, etc. Bref, une rupture totale avec la précédente lignée et cela se vérifie par la non-compatibilité des applications Windows Mobile 6.5 avec le Phone 7 ! Ce qui va nécessiter une refonte du code, de l'interface. De longues heures d'adaptation en perspective.

Cette rupture était cependant inévitable pour repartir sur des bases saines et stables. C'est aussi pour cette raison que Microsoft a dévoilé très tôt les outils de développement autour de Visual Studio for Windows Phone 7 afin que les développeurs puissent rapidement tester la plateforme et commencer à adapter le code source et à produire les nouvelles applications. Notons que pour les constructeurs, le cahier des charges exigé par Microsoft est très strict. L'objectif est de limiter une disparition entre les différents mobiles



Architecture système et applicative Windows.



et éviter la profusion de SDK, de fonctions exotiques et non-standard. Pour le développeur, cela devrait garantir une meilleure compatibilité et réduire les mauvaises surprises et pour l'utilisateur, avoir une utilisation la plus transparente possible quel que soit le terminal.

Côté développement, Windows Phone 7 s'appuie sur 3 piliers : Silverlight, Visual Studio 2010 / .Net et XNA (pour les jeux). Il introduit aussi un nouveau rôle au designer d'interface avec Expression Blend qui devient l'IDE graphique par excellence pour concevoir les interfaces, le design d'interface. Pour débuter en Phone 7, il faut télécharger Windows Phone Developer Tools (à l'heure ou nous écrivons toujours en version technique CTP). Il inclut Visual Studio Express, l'émulateur, Silverlight pour Windows Phone, XNA Game Studio.

La soumission des applications au Market Place a été une épreuve de force et Microsoft a révisé plusieurs fois sa position. Actuellement, les modalités pour soumettre une application sont les suivantes :

- payer un abonnement de 99 $ US annuel pour soumettre des applications gratuites (5 par an, puis paiement pour chaque application gratuite supplémentaire), pas de limite pour les logiciels payants
- disponibilité d'une notification push
- pas de limitation géographique
- 70 % des ventes pour le développeur, 30 % pour Microsoft
- validation de l'application par Microsoft
- gratuité pour les étudiants (Dreamsparck)

Un marché en devenir sans aucun doute et des opportunités pour les développeurs qui voudraient élargir l'horizon technologique…

Site : http://developer.windowsphone.com

Android

Editeur / constructeur : *Google / divers*
Système : *Android 2.x*
IDE principal : *Eclipse*
Langage natif principal : *Java*
Nb applications sur la boutique en ligne : *+ 50 000*

Android est le projet de système mobile basé sur un cœur Linux et utilisant, comme d'autres systèmes, WebKit pour l'affichage des pages web. Il est initié par Google et connaît une croissance très forte sur le marché autant en nombre de terminaux disponibles que d'applications et d'utilisateurs. Le langage principal est Java et utilise l'IDE Eclipse.

Le développeur dispose des éléments suivants :

- Android SDK : c'est le cœur du développement Android, supporte les 3 systèmes desktop principaux.
- Android Development Tools (ADT) : plug-in Eclipse pour construire et builder les projets Android.
- SDK Tools : composant pour le Android SDK. Il contient un ensemble d'outils pour développer et déboguer les applications.
- Pour des développements C ou C++, il faudra passer par le NDK, kit de développement natif.

Télécharger les outils : http://developer.android.com/sdk/index.html

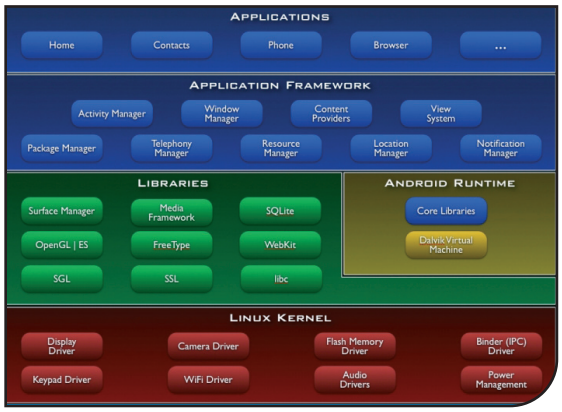
A noter qu'il est possible de développer depuis NetBeans pour Android (projet nbandroid). Cependant, le projet NetBeans évolue peu.

On évoque souvent la fragmentation du marché avec la multiplication des terminaux Android ainsi que la multiplication des versions du système dans la nature, ce qui peut obliger à choisir une version plus ancienne que celle actuellement disponible afin d'être présent sur une majorité des terminaux.

Faux problème pour les uns, vrais problèmes pour les autres. Quoi qu'il en soit, rien ne vaut des tests sur les terminaux physiques et non pas uniquement sur l'émulateur même si les projets sont de bonne qualité.

Un des forces d'Android est la grande liberté laissée aux développeurs pour publier une application.

Nous n'avons pas les contraintes d'un Apple pour la validation iPhone / iPad. Il est tout de même nécessaire de respecter les règles élémentaires pour builder une bonne application Android : avoir un contrat d'utilisation, pas de mode debug, obtenir une clé cryptographique, signature de l'application… Il faut avoir un compte sur le Android Market avant toute publication : 25 $ US et accepter les



conditions générales de distribution. Guide du développement : http://developer.android.com/guide/index.html

Et les plates-formes AIR et Flash ?

Aujourd'hui, Adobe veut être un acteur du développement mobile en proposant une plate-forme unique pour plusieurs systèmes mobiles, que ce soit en Flash 10.1 ou en Air 2. Actuellement Android est le plus en avance, même si Air n'est qu'en bêta et que le player Flash 10.1 attend les terminaux mobiles sur le marché pour réellement prendre son envol. Adobe travaille aussi avec Nokia mais l'agenda n'est pas connu.

Sur Blackberry, le player 10.1 sera disponible au moins en pré-version dans les prochains mois.

Et sur Windows Phone 7, une version sera disponible dans les mois qui suivront la sortie effective de la plate-forme.

Sur iPhone, Adobe confirme l'arrêt de tout développement par rapport à Flash et Air, même si le iPhone Packager est disponible gratuitement pour bâtir des projets internes sur la plate-forme mobile Apple. Il faudra tout de même attendre 2011 avant que le développeur ne dispose de l'ensemble des outils et frameworks pour lui faciliter la vie, notamment sur tout ce qui concerne l'adaptation automatique des interfaces selon la taille et l'orientation de l'écran…

■ François **Tonic**

Créer une application universelle pour iPad

Beaucoup d'acteurs se sont laissés surprendre par le succès de l'iPhone, puis de l'App Store. Aujourd'hui, il est devenu très difficile de se faire remarquer parmi plus de 200 000 applications proposées. De ce point de vue, l'iPad représente ainsi une seconde opportunité d'accéder à la visibilité de l'App Store. Tout d'abord parce que le nombre d'applications disponibles est actuellement beaucoup plus faible (plus de 20 000 au moment de la rédaction de cet article) et aussi parce que cet appareil présente des caractéristiques susceptibles de correspondre à de nouveaux usages.

Ces quelques pages vous proposent donc de mettre en œuvre les spécificités de l'iPad tout en créant une application fonctionnant également sur l'iPhone, avec les mêmes outils, le même langage.

L'iPad vu par le développeur

L'iPad est parfois décrit comme un gros iPod Touch et c'est vrai que cette tablette reprend l'ensemble des éléments qui ont fait le succès des précédents périphériques tactiles d'Apple, aussi bien auprès des utilisateurs que des développeurs :

- L'utilisateur retrouve la même interface tactile et peut même faire fonctionner les applications conçues pour l'iPhone ou l'iPod Touch.
- Le développeur bénéficie de l'environnement Cocoa/Xcode/Interface Builder, mis au point par la société NeXT (également fondée par Steve Jobs) à la fin des années 1980 puis affinés depuis par Apple. Cet environnement repose sur le langage Objective-C et sur des design patterns éprouvés par de multiples évolutions. Tout cela donne un mélange incroyablement efficace en combinant innovation, modernité et expérience longue d'une vingtaine d'années. En plus de tous ces éléments communs avec l'iPhone, l'iPad présente deux caractéristiques qui le distinguent : les dimensions de l'écran et le fait de l'utiliser dans n'importe quelle orientation. Ces nouvelles possibilités complètent les règles de navigation dans l'information jusque-là proposées par le système.

UIViewController

La classe UIViewController est le véritable chef d'orchestre de l'interface utilisateur. C'est elle qui définit, pour les principaux contextes de l'application, le lien entre les vues (l'interface graphique) et le modèle de données sous-jacent.
La conception de l'application consiste donc à définir les vues principales et à créer les contrôleurs associés.
Un UIViewController est très souvent associé à un fichier XIB. Ce fichier, dessiné dans l'outil graphique nommé Interface Builder, contient la vue correspondante, ainsi que toutes les sous-vues qui sont sous son contrôle. Ce fichier XIB définit également les liens entre les objets graphiques et les méthodes correspondantes du contrôleur.

Les règles de navigation

Ces règles ont été définies pour manipuler une information très dense avec des gestes très imprécis en comparaison de la taille des pixels affichés. La conception d'une application pour iOS commence donc par une transformation de l'ensemble des informations en une succession d'écrans élémentaires.
La réduction du nombre d'informations affichées simultanément est d'ailleurs un gage de simplicité pour l'utilisateur : en effet, le développeur doit épurer son œuvre pour ne présenter que l'essentiel. Un écran débarrassé de nombreuses fioritures est ainsi

moins complexe à utiliser, mais il est aussi plus difficile à mettre au point ! C'est pourquoi, Apple a déjà imaginé ces grandes règles permettant d'assembler les écrans entre eux. Faire appel à ces assemblages préconçus est particulièrement avantageux :

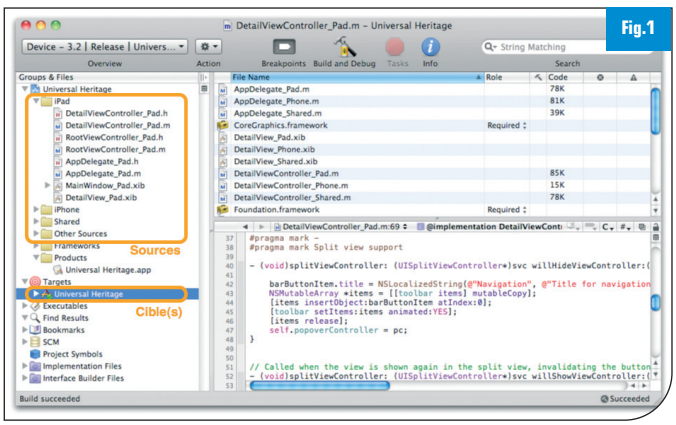
- Le développeur peut s'appuyer sur des mécanismes éprouvés et dont les composants sont déjà prêts à l'emploi. Il se consacre alors à l'optimisation des fonctions propres à l'application.
- L'utilisateur acquiert des réflexes communs pour la majorité des applications. Ces réflexes permettent à l'utilisateur de se concentrer sur le contenu de la tâche à accomplir. Ils rendent ainsi la navigation plus naturelle et plus intuitive.

Le tableau A indique chacune des formes initiales pour assembler les écrans de l'application et indique la classe qui l'implémente.

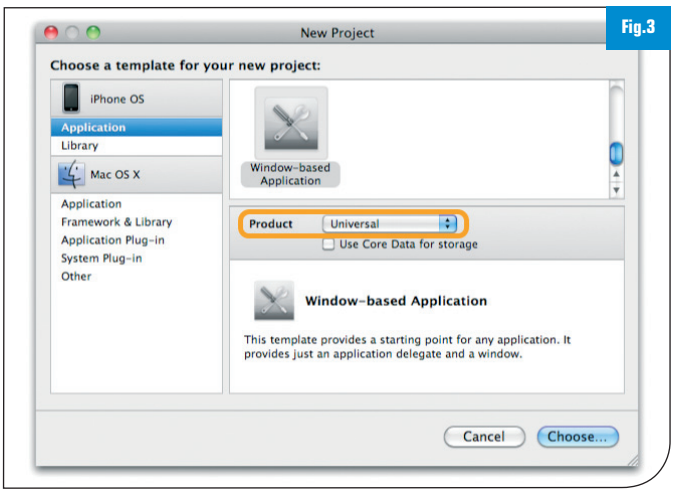
Tableau A — Règles de navigation de base

Règle de navigation	Classe correspondante
Défilement, éventuellement accompagné d'un facteur de grossissement	UIScrollView
Enchaînement modal qui remplace (temporairement) un écran par un autre	UIViewController
Segmentation parallèle qui filtre l'affichage de l'information	UITabBar
Navigation hiérarchique qui correspond à une exploration arborescente de l'information	UINavigationController

L'iPad peut mettre en œuvre deux formes supplémentaires qui profitent de la taille de l'écran.



Le projet Xcode



Création d'application

Tableau B — Règles de navigation propres à l'iPad

Règle de navigation	Classe correspondante
Superposition partielle (et temporaire) d'une vue sur une autre	UIPopoverController
Juxtaposition de deux vues	UISplitViewController

Construction et mise en ligne

Une application iOS est construite à partir d'un projet Xcode. Le projet rassemble l'ensemble des fichiers sources et des ressources qui sont mis en œuvre par une « cible » pour produire l'application. Un projet peut ainsi contenir plusieurs cibles : par exemple une cible iPhone et une cible iPad [Fig.1].

Chaque cible correspond généralement à une application qui pourra être mise en ligne sur l'App Store au travers d'un compte sur iTunes Connect, créé par le développeur, et de la section « Archived Applications » de l'organiseur proposé par Xcode [Fig.2].

Types d'application iOS

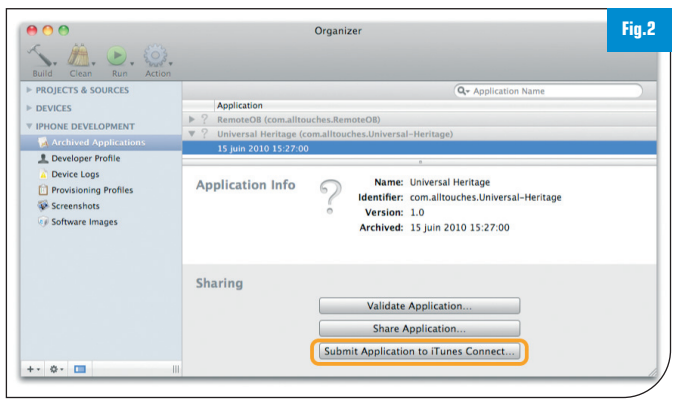
Un projet peut ainsi produire :

- une application pour l'iPhone ;
- une application pour l'iPad ;
- une application « universelle » qui fonctionne à la fois sur iPhone et sur iPad.

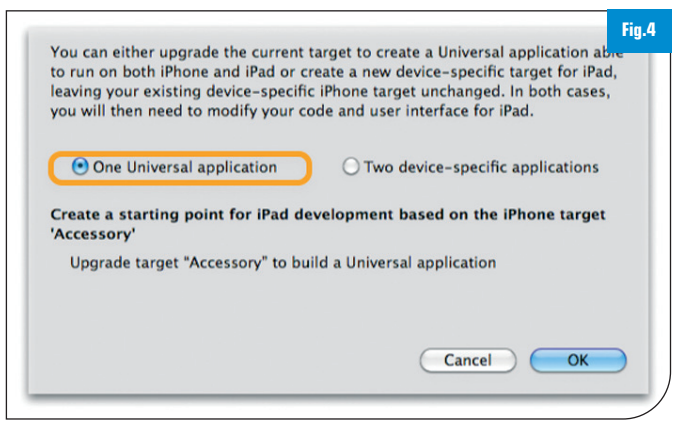
Le type de l'application produite par la cible est habituellement fixé à la création d'un nouveau projet (voir Fig.3.) ou bien obtenu en migrant un projet iPhone (voir Fig.4.).

Apple recommande de créer une application universelle parce qu'elle présente des attraits particuliers pour l'utilisateur :

Avec une seule visite sur l'App Store, celui-ci peut exploiter des



L'organiseur de Xcode



Migration d'application

fonctionnalités communes quel que soit son périphérique. Les interactions peuvent, bien entendu, être différentes selon l'appareil mais le cœur de l'application demeure identique.
En cas de changement de périphérique, l'ensemble des données de l'application est maintenu, permettant à l'utilisateur de retrouver ses réalisations précédentes. Le développeur bénéficie également d'avantages appréciables : son application est mise en valeur avec un signe « + » aussi bien dans la liste des applications pour iPhone que dans la liste des applications pour iPad. La mise en ligne et la maintenance sont consacrées à un binaire unique, ce qui divise par deux la charge administrative !

Les classes de l'application

Un projet Xcode créé ou migré avec une cible d'application universelle propose la juxtaposition basique de deux ensembles de classes :

- Les classes `AppDelegate_Phone`, `RootViewController_Phone` et `DetailViewController_Phone` pour l'application iPhone ;
- Les classes `AppDelegate_Pad`, `RootViewController_Pad` et `DetailViewController_Pad` pour l'application iPad.

La classe `AppDelegate_Phone`, produite par Xcode, ne partage aucun élément avec la classe `AppDelegate_Pad`. De même, les contrôleurs pour l'iPhone ou pour l'iPad sont totalement distincts. On peut cependant supposer, de manière légitime, que les instructions décrivant une même application pour différents périphériques comportent de nombreuses fonctions communes. Nous allons donc modifier ce cadre initial pour favoriser la conception commune de la version iPhone et de la version iPad d'une application.La solution adoptée consiste à placer les variables et les méthodes communes à l'iPhone et à l'iPad dans un ascendant commun dont le suffixe est `_Shared` [Fig.5].

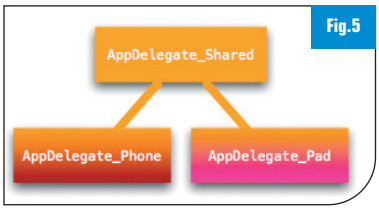


Fig.5
Ascendant commun

Les descendants `_Phone` et `_Pad` se contentent alors de gérer les comportements propres à chaque périphérique. Il peut s'agir :

- soit de fonctionnalités définies spécifiquement pour un appareil, telles que la juxtaposition de vues pour l'iPad ;
- soit d'ajustements du comportement général qui est récupéré par héritage. C'est, par exemple, le cas de l'initialisation du délégué de l'application :

```
- (BOOL)application:(UIApplication *)application
             didFinishLaunchingWithOptions:(NSDictionary
 *)launchOptions {

  // Initialisation propre au périphérique
  …
  // Initialisation commune
  return [super application:application
             didFinishLaunchingWithOptions:launchOptions];
}
```

Pour faire évoluer l'application, aussi bien dans sa version iPhone que dans sa version iPad, il suffit alors de modifier l'ascendant commun.

iOS

iOS est le nouveau nom d'iPhone OS, le système d'exploitation conçu pour les périphériques tactiles Apple. Ce système est une évolution de Mac OS X, lui-même provenant de NeXTSTEP. Ce système est un UNIX, complété par des services tels que le moteur graphique, par-dessus lequel tourne l'environnement objet Cocoa Touch. Cet environnement Objective-C comporte deux frameworks principaux : Foundation et UIKit. La figure 8 illustre cette architecture ainsi que les principales conventions de nommage de ce système.

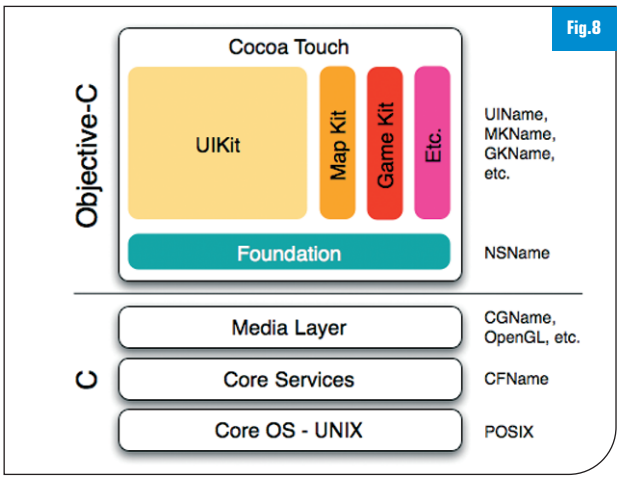


Fig.8
Figure 8 — l'architecture d'iOS

Foundation est composé des classes fondamentales représentant des notions telles que les listes, les chaînes de caractères, les objets persistants, etc. Depuis la version 4 d'iOS, *Foundation* reprend désormais l'essentiel des éléments de Mac OS X Snow Leopard, avec, en particulier la notion de bloc d'exécution et la gestion d'exécutions parallèles avec *Grand Central Dispatch*. *UIKit* propose l'ensemble des classes applicatives permettant de créer une application tactile. Les *design patterns* et les assemblages par défaut générés par les outils de développement assurent un fonctionnement à la fois robuste et évolutif. La version 4 d'iOS est actuellement réservée à l'iPhone (sortie : 21 juin dernier), la version 4 pour iPad est annoncée pour cet automne.

Fig.6
Contrôleur commun de navigation

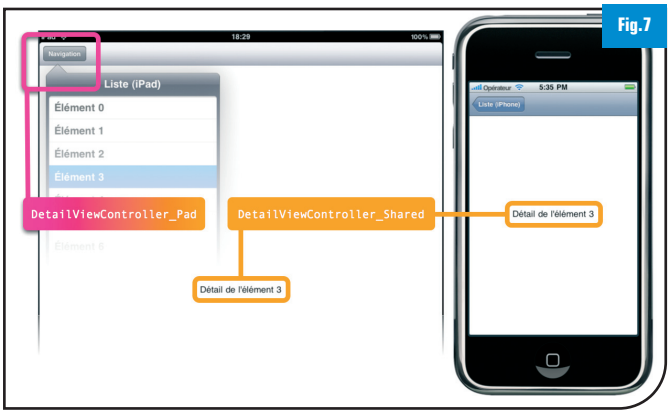
Navigation avec l'iPad et l'iPhone

L'iPad propose l'affichage de vues juxtaposées, comme indiqué dans le tableau B. Ces deux vues sont, le plus souvent, une vue de navigation et une vue présentant les détails de l'élément sélectionné par la navigation.

Le contenu de ces mêmes vues peut être présenté avec une navigation hiérarchique sur un iPhone : lorsque l'utilisateur sélectionne un élément, la vue de détail correspondante est poussée sur l'écran.

Puisque le contenu est commun, ce type de présentation est le candidat idéal pour illustrer l'héritage de fonctionnalité ! [Fig.6].

Fig.7

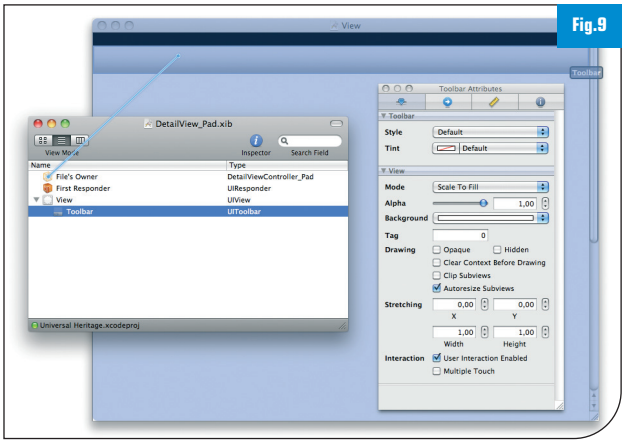


Contrôleur commun de la vue de détail

Le SDK

L'ensemble des ressources nécessaires pour développer une application pour iOS est proposé par le SDK (Software Development Kit) disponible sur le site http://developer.apple.com/. Ce kit comprend les outils fondamentaux : Xcode, Interface Builder et le simulateur de périphérique (iPhone ou iPad), ainsi que des outils de mise au point comme *Instruments*. Il comprend également la documentation complète et des exemples de code. Pour télécharger le SDK, vous devez créer un compte (gratuit) auprès d'Apple. Pour signer une application sur votre périphérique ou sur l'App Store, vous devez compléter ce compte avec une souscription annuelle s'élevant à 79 €. Les outils nécessitent un Mac avec le système Mac OS X version 10.6.

Fig.9



Interface Builder

La navigation commune est ici réalisée par la classe `RootViewController_Shared` qui joue, à la fois, le rôle de `UITableViewController`, `UITableViewDataSource` et `UITableViewDelegate` pour la liste des éléments. La classe `RootViewController_Phone` ajoute l'accessoire `DisclosureIndicator` :

```
- (UITableViewCell *)tableView:(UITableView *)tableView
     cellForRowAtIndexPath:(NSIndexPath *)indexPath {

  UITableViewCell *cell = [super tableView:tableView
                 cellForRowAtIndexPath:indexPath];
   cell.accessoryType  =  UITableViewCellAccessoryDisclosure
Indicator;
  return cell;
}
```

Un tel accessoire signale la présence de la navigation hiérarchique. La vue de détail est poussée grâce à un appel à la méthode `pushViewController` :

```
- (void)tableView:(UITableView *)aTableView
       didSelectRowAtIndexPath:(NSIndexPath *)indexPath {

   [self.navigationController pushViewController:detailView
Controller
             animated:YES];

   [super tableView:aTableView didSelectRowAtIndexPath:indexPath];
}
```

La classe `RootViewController_Pad` ne présente pas d'accessoire et laisse tout simplement l'ascendant commun modifier la vue de détail lorsqu'une ligne est sélectionnée.

Cette vue de détail est gérée par la classe `DetailViewController_Shared`. La version spécialisée pour iPad de cette classe joue, en plus, le rôle de `UIPopoverControllerDelegate` et de `UISplitViewControllerDelegate`. Ces deux protocoles assurent la prise en compte de la navigation dans une vue *popover* lorsque l'iPad est placé en mode portrait (voir Fig.7).

Gestion de l'orientation

Chaque contrôleur de vue principale peut déterminer une orientation d'affichage. Apple recommande qu'une application pour iPad gère toutes les orientations possibles. La prise en compte d'une orientation est signalée par la méthode shouldAutorotateToInterfaceOrientation: du contrôleur. Il suffit de retourner YES en fonction du paramètre passé.

On peut également souhaiter offrir un comportement totalement différent en fonction de l'orientation du périphérique. Par exemple, en mode paysage, l'usage du clavier est plus facile. Au contraire, les listes sont plus agréables en mode portrait. La classe UISplitViewController masque ainsi la vue de navigation lorsque l'iPad est en mode portrait. Cette vue reste accessible au travers d'un bouton qui la présente en mode popover (voir figure 4).

Le code de mise en œuvre

L'ensemble du code source de cet exemple constitue le projet `Universal Heritage` disponible sur le site http://alltouches.com/. Il combine trois des principaux modèles de projets **Xcode** :

- *(Universal) Window-based Application* pour l'architecture générale d'une application universelle.
- *(iPhone) Navigation-based Application* pour la navigation hiérarchique à partir d'une liste.
- *(iPad) Split View-based Application* pour la gestion des vues juxtaposées.

La classe `DetailViewController_Pad` implémente l'essentiel des spécificités de l'iPad avec le code provenant du modèle *Split View-based Application*.

Le rôle de délégué pour la gestion des vues juxtaposées suppose, en particulier, d'ajouter ou de supprimer le bouton appelant la vue de navigation lors du changement d'orientation de l'iPad. Une première méthode du délégué est invoquée lorsque la vue de navigation va être masquée :

```
- (void)splitViewController: (UISplitViewController*)svc
     willHideViewController:(UIViewController *)aViewController
     withBarButtonItem:(UIBarButtonItem*)barButtonItem
   forPopoverController: (UIPopoverController*)pc {

 barButtonItem.title = NSLocalizedString(@»Navigation», @»Title
for navigation button»);
 NSMutableArray *items = [[toolbar items] mutableCopy];
 [items insertObject:barButtonItem atIndex:0];
 [toolbar setItems:items animated:YES];
 [items release];
 self.popoverController = pc;
}
```

Cette méthode fixe l'intitulé du bouton passé en paramètre puis insère ce bouton dans la barre d'outils gérée par ce contrôleur. Cette barre est référencée par un lien créé par **Interface Builder** dans le fichier XIB associé au contrôleur (voir Fig.9).

La méthode complémentaire est appelée lorsque la vue de navigation va être juxtaposée :

```
- (void)splitViewController:(UISplitViewController*)svc
   willShowViewController:(UIViewController *)aViewController
 invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem {

   NSMutableArray *items = [[toolbar items] mutableCopy];
   [items removeObjectAtIndex:0];
   [toolbar setItems:items animated:YES];
   [items release];
   self.popoverController = nil;
}
```

Cette méthode réalise l'opération inverse en retirant le bouton de la barre d'outils.

On peut également remarquer que le contrôleur conserve une référence vers le gestionnaire de vue superposée (`popoverController`) afin de pouvoir fermer cette vue lorsque la navigation est terminée :

```
- (void)setDetailItem:(id)newDetailItem {

  [super setDetailItem:newDetailItem];

  if (popoverController != nil) {
    [popoverController dismissPopoverAnimated:YES];
  }
}
```

La dernière méthode que nous détaillons est celle qui permet d'ajuster les dimensions de la vue de détail à celle de l'iPad. En effet, la vue de détail est décrite par un fichier d'interface, nommé `DetailView_Shared.xib`, commun à la version iPhone et à la version iPad :

```
- (void)viewDidLoad {
  [super viewDidLoad];
  [[NSBundle mainBundle] loadNibNamed:@»DetailView_Shared» owner:
self options:nil];
  detailView.frame = self.detailViewFrame;
  [self.view addSubview:detailView];
  self.detailView = nil;
}
```

La propriété `detailView` est automatiquement fixée lors de la désérialisation du fichier `DetailView_Shared.xib`. Les dimensions de la vue sont fournies par l'accesseur detailViewFrame. Une fois la vue correspondante insérée dans la hiérarchie, on libère cette propriété pour que la vue ne soit plus maintenue en mémoire que par sa présence dans la hiérarchie. L'ancêtre commun étend la vue de détail à la surface totale gérée par le contrôleur associé :

```
- (CGRect)detailViewFrame {
  CGSize frameSize = self.view.frame.size;
  return CGRectMake(0.0, 0.0, frameSize.width,
          frameSize.height);
}
```

Le contrôleur spécifique à l'iPad surcharge l'accesseur detailViewFrame afin de prendre en compte la barre d'outils :

```
- (CGRect)detailViewFrame {
  CGSize frameSize = self.view.frame.size;
  CGFloat toolBarHeight = toolbar.frame.size.height;
  return CGRectMake(0.0, toolBarHeight,
       frameSize.width, frameSize.height - toolBarHeight);
}
```

Ce dernier exemple illustre comment la programmation-objet qui sous-tend cette solution permet de ne décrire que les différences propres à chaque périphérique.

■ Etienne **Vautherin**
Facilite la création innovante aux Orange Labs, il est également l'auteur de l'ouvrage « Développer pour l'iPhone et l'iPad » chez Dunod. Site : http://alltouches.com/

Développer sa première application Windows Phone 7

Avec Windows Phone 7, Microsoft a décidé de repartir de zéro pour tout ce qui concerne la mobilité. Nous allons voir ce que cela signifie pour nous les développeurs d'applications mobiles.

Présentation de Windows Phone 7

Avec Windows Phone 7, ce n'est pas simplement le système d'exploitation qui a changé mais aussi et surtout le matériel. Ainsi, Microsoft impose que celui-ci comporte un processeur ARMv7, un GPU qui supporte DirectX (afin de pouvoir être exploité, au mieux, par XNA pour tout ce qui concerne les applications de jeux vidéo), ainsi que toute une batterie de capteurs : GPS, luminosité, boussole, etc.

L'une des grosses problématiques, pour les développeurs sous Windows Mobile, était de devoir gérer plusieurs résolutions d'écrans. Avec Windows Phone 7, Microsoft simplifie énormément les choses car on ne dispose plus que de deux résolutions : WVGA (800x480) et HVGA (480x320). De plus, afin de pouvoir offrir des applications d'un nouveau type, le matériel embarque un écran capacitif supportant, au moins, 4 points de contact !

Le système d'exploitation, basé sur une Windows Embedded CE 6, a lui aussi été entièrement repensé pour faire en sorte que Microsoft soit en charge de la plus grande partie du code [Fig.1].

Ainsi, la seule partie que les fabricants de matériel sont en mesure de modifier est la couche basse des drivers. L'objectif de ce « bridage » est de permettre la mise à disposition d'une API unifiée pour les capteurs (GPS, etc.) mais également pour les couches graphiques, multimédia, etc. Pour les développeurs, il faut savoir que même la partie développement a changé. En effet, sur les précédentes versions de Windows Mobile, on utilisait le Compact Framework pour développer des applications mobiles. Avec Windows Phone 7, il est possible d'utiliser 2 autres technologies : XNA ou Silverlight (technologie utilisée dans le reste de l'article). Cependant, étant donné que rien n'est installé directement, il est nécessaire de mettre en place son environnement de développement pour pouvoir commencer à travailler.

Installer les outils de développement

Les outils de développement, pour Visual Studio 2010 et que l'on utilise pour développer avec Silverlight ou XNA, sont disponibles, gratuitement, à l'adresse suivante : http://msdn.microsoft.com/fr-fr/windowsphone/ff727985.aspx. Pour le moment, ces outils sont en version CTP (*Community Technology Preview*), ce qui signifie qu'ils ne sont pas encore complètement finalisés (ce qui est plutôt logique lorsque l'on réfléchit bien, étant donné que le système d'exploitation lui-même n'est pas terminé non plus).

Une fois installé, vous disposez d'une version Express de Visual Studio 2010, spécialement conçue pour le développement Windows Phone 7 : [Fig.2].

Cependant, si vous disposez déjà d'un Visual Studio 2010 (Express, Standard, etc.), alors les templates de projets nécessaires à la création de projets Windows Phone 7 sont installés directement sur votre ordinateur : [Fig.3].

Fig.2

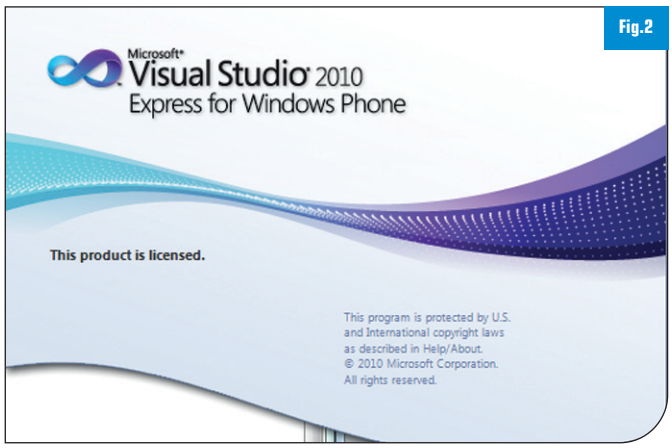


Fig.1

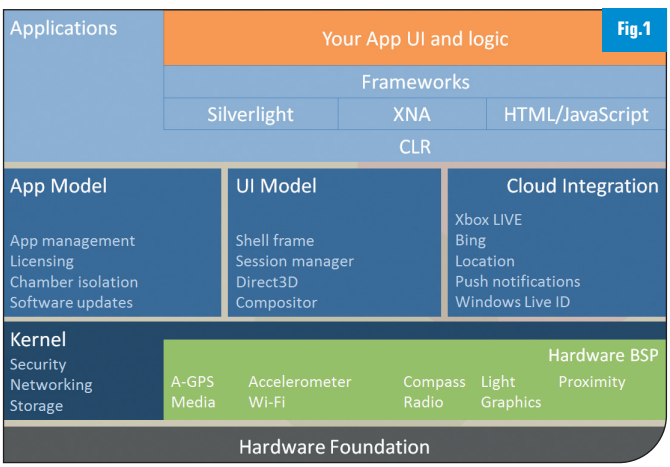
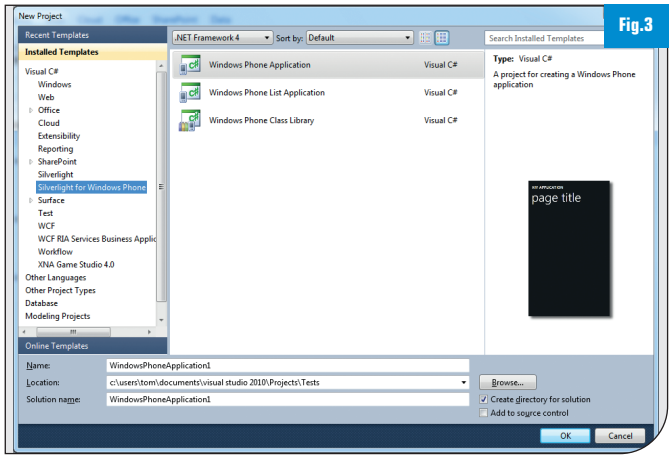


Fig.3



Développer sa première application

Pour développer sa première application Windows Phone 7, c'est très simple : lancer la fenêtre de nouveau projet, sélectionnez « Windows Phone Application » et cliquez sur « OK ».

À ce moment, le designer de Visual Studio va s'afficher et vous présenter une interface séparée en deux parties :

- Sur la gauche, la représentation graphique de votre application, autrement dit la manière dont elle va être représentée une fois l'application exécutée
- Sur la droite, le code XAML servant à décrire l'interface graphique [Fig.4].

On remarque immédiatement que sans avoir eu besoin de faire quoi que ce soit, l'interface graphique dispose déjà d'un thème pour l'application. En effet, si vous jetez un coup d'œil au code XAML qui a été généré, on remarque qu'il fait appel à des styles :

```
<Grid x:Name=»LayoutRoot» Background=»{StaticResource PhoneBackgroundBrush}»>
  <Grid.RowDefinitions>
    <RowDefinition Height=»Auto»/>
    <RowDefinition Height=»*»/>
  </Grid.RowDefinitions>

  <!—TitleGrid is the name of the application and page title—>
    <Grid x:Name=»TitleGrid» Grid.Row=»0»>
      <TextBlock Text=»MY APPLICATION» x:Name=»textBlockPageTitle» Style=»{StaticResource PhoneTextPageTitle1Style}»/>
      <TextBlock Text=»page title» x:Name=»textBlockListTitle» Style=»{StaticResource PhoneTextPageTitle2Style}»/>
    </Grid>

    <!—ContentGrid is empty. Place new content here—>
    <Grid x:Name=»ContentGrid» Grid.Row=»1»>
  </Grid>
</Grid>
```

Ces styles proviennent tout simplement du fichier App.xaml (fichier central/commun à tous les éléments de l'application), qui en définit une très grande quantité (en voici un aperçu) :

```
<!—************ THEME RESOURCES ************—>
<!— Color Resources —>
<Color x:Key=»PhoneBackgroundColor»>#FF1F1F1F</Color>
<Color x:Key=»PhoneContrastForegroundColor»>Black</Color>
<Color x:Key=»PhoneForegroundColor»>White</Color>
<Color x:Key=»PhoneInactiveColor»>#FF666666</Color>
<Color x:Key=»PhoneDisabledColor»>#FF808080</Color>
<Color x:Key=»PhoneSubtleColor»>#FF999999</Color>
<Color x:Key=»PhoneContrastBackgroundColor»>#FFFFFFFF</Color>
<Color x:Key=»PhoneTextBoxColor»>#FFBFBFBF</Color>
<Color x:Key=»PhoneBorderColor»>#FFCCCCCC</Color>
<Color x:Key=»PhoneTextSelectionColor»>Black</Color>
<Color x:Key=»PhoneAccentColor»>#FF1BA1E2</Color>

<!— Brush Resources —>
<SolidColorBrush x:Key=»PhoneAccentBrush» Color=»{StaticResource PhoneAccentColor}»/>
<SolidColorBrush x:Key=»PhoneBackgroundBrush» Color=»{StaticResource PhoneBackgroundColor}»/>
<SolidColorBrush x:Key=»PhoneContrastForegroundBrush» Color=»{StaticResource PhoneContrastForegroundColor}»/>
<SolidColorBrush x:Key=»PhoneForegroundBrush» Color=»{StaticResource PhoneForegroundColor}»/>
<SolidColorBrush x:Key=»PhoneInactiveBrush» Color=»{StaticResource PhoneInactiveColor}»/>
<SolidColorBrush x:Key=»PhoneDisabledBrush» Color=»{StaticResource PhoneDisabledColor}»/>
<SolidColorBrush x:Key=»PhoneSubtleBrush» Color=»{StaticResource PhoneSubtleColor}»/>
<SolidColorBrush x:Key=»PhoneContrastBackgroundBrush» Color=»{StaticResource PhoneContrastBackgroundColor}»/>
<SolidColorBrush x:Key=»PhoneTextBoxBrush» Color=»{StaticResource PhoneTextBoxColor}»/>
<SolidColorBrush x:Key=»PhoneBorderBrush» Color=»{StaticResource PhoneBorderColor}»/>
<SolidColorBrush x:Key=»PhoneTextSelectionBrush» Color=»{StaticResource PhoneTextSelectionColor}»/>
<SolidColorBrush x:Key=»TransparentBrush» Color=»Transparent»/>

<!— Touch Target Area —>
<Thickness x:Key=»PhoneTouchTargetOverhang»>12</Thickness>
```

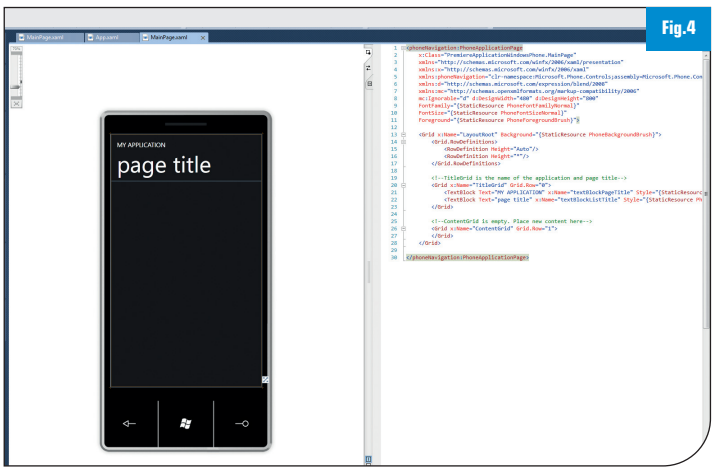


Fig.4

Ainsi, si vous souhaitez personnaliser le thème de votre application, nul besoin de modifier chacun des éléments se trouvant dans un fichier XAML : vous modifiez directement la valeur dans le fichier App.xaml et cette modification sera répercutée automatiquement sur chacun des éléments qui utilisaient cette ressource.

Pour commencer le développement de votre application, il vous suffit de faire glisser/déposer les contrôles qui vous intéressent directement depuis la boîte à outils sur la surface de dessin.

Ainsi, le code suivant :

```
<!—TitleGrid is the name of the application and page title—>
  <Grid x:Name=»TitleGrid» Grid.Row=»0»>
    <TextBlock Text=»Ma première application Windows Phone 7» x:Name=»textBlockPageTitle» Style=»{StaticResource PhoneTextPageTitle1Style}»/>
    <TextBlock Text=»Programmez !» x:Name=»textBlockListTitle» Style=»{StaticResource PhoneTextPageTitle2Style}»/>
  </Grid>
```

```
<!—ContentGrid is empty. Place new content here—>
<Grid x:Name=»ContentGrid» Grid.Row=»1»>
  <ListBox Height=»569»
    HorizontalAlignment=»Left»
    Margin=»6,0,0,0»
    Name=»listBox1»
    VerticalAlignment=»Top»
    Width=»474» />
  <Button Content=»Show Image»
    Height=»70»
    Margin=»141,574,140,0»
    Name=»button1»
    VerticalAlignment=»Top» />
</Grid>
```

Permet d'afficher un contrôle de type ListBox avec un bouton juste en dessous, comme vous pouvez le constater sur l'image [Fig.5].

Bien sûr, l'application étant une application Silverlight 3, il est possible de lui rajouter de la logique, du comportement pour la rendre plus interactive. Pour cela, sur le bouton, abonnez-vous à l'événement Click !

```
<Button Content=»Show Image»
   Height=»70»
   Margin=»141,574,140,0»
   Name=»button1»
   VerticalAlignment=»Top»
   Click=»button1_Click»/>
```

Puis, dans le code behind de la page, vous pouvez rajouter la logique faisant le lien avec les actions de l'utilisateur :

```
private void button1_Click(object sender, RoutedEventArgs e)
{
 string logoUrl = «http://www.programmez.com/img/logo.jpg»;

 WebClient webClient = new WebClient();
 webClient.OpenReadCompleted += (o, args) =>
 {
   Image image = new Image();

   ListBoxItem listBoxItem = new ListBoxItem();
   listBoxItem.Content = image;
   listBoxItem.Tag = «http://www.programmez.com»;

   BitmapImage bitmapImage = new BitmapImage();
   bitmapImage.SetSource(args.Result);

   image.Source = bitmapImage;

   this.listBox1.Items.Add(listBoxItem);
 };
 webClient.OpenReadAsync(new Uri(logoUrl));
}
```

Ainsi, le code suivant permet d'appeler l'URL du logo du magazine Programmez, de la télécharger et de l'afficher dans un nouvel élément de la ListBox, comme on peut le vérifier sur l'image : [Fig.6].
Les applications Windows Phone 7 disposent d'un framework de navigation très puissant, qui leur permet de naviguer de page en page. Pour cela, il est d'abord nécessaire de rajouter une nouvelle page à son application, en faisant un clic droit sur le nom du projet puis en sélectionnant « *Add* » => « *New Item* » : [Fig.7].
Dans la fenêtre qui s'ouvre, choisissez « Silverlight for Windows Phone », sélectionnez l'une des options disponibles et cliquez sur « *Add* » :

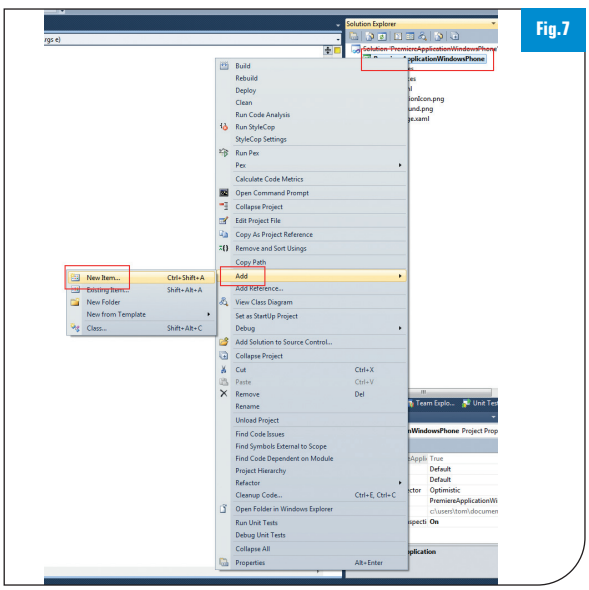
- Windows Phone Landscape Page : Permet d'ajouter une nouvelle page qui sera affichée en mode paysage
- Windows Phone Portrait Page : Permet d'ajouter une nouvelle page qui sera affichée en mode portrait
- Windows Phone User Control : Permet d'ajouter un nouveau contrôle utilisateur [Fig.8].

Il suffit maintenant d'indiquer à notre application que l'on souhaite naviguer vers cette page, lorsque l'utilisateur sélectionne un élément dans la ListBox. Pour cela, on va simplement s'abonner à l'évènement SelectionChanged de la ListBox et récupérer l'item sélectionné :

Fig.5

Fig.6

Fig.7



```
private void listBox1_SelectionChanged(object sender, Selection
ChangedEventArgs e)
{
 ListBoxItem listBoxItem = e.AddedItems[0] as ListBoxItem;
 if(listBoxItem != null)
 {
   string url = string.Format(«/Page1.xaml?site={0}», listBox
Item.Tag);

   NavigationService.Navigate(new Uri(url, UriKind.Relative));
 }
}
```

Vous pouvez voir qu'on utilise la méthode *Navigate*, de la classe NavigationService, pour se diriger vers une autre page de l'application, en lui passant éventuellement des paramètres dans l'URL. Dès lors, il devient intéressant de se demander comment récupérer ces paramètres. Et pour cela, nous allons surcharger, sur notre nouvelle page, la méthode OnNavigateTo et faire appel à la classe *NavigationContext* :

```
protected override void OnNavigatedTo(Microsoft.Phone.Naviga
tion.PhoneNavigationEventArgs e)
{
 base.OnNavigatedTo(e);

 string url = NavigationContext.QueryString[«site»];
```

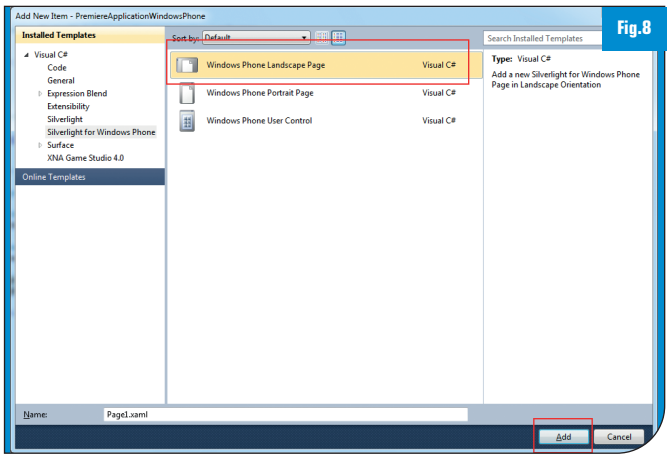


Fig.8

```
 NavigationService.Navigate(new Uri(url));
}
```

Vous pouvez ainsi voir que le service de navigation peut être utilisé pour naviguer entre les pages de l'application mais également vers d'autres pages Internet !

L'expérience de débogage

Comme toute technologie, le développeur en charge de réaliser l'application n'est pas en mesure de garantir un code 100 % parfait. Ainsi, il lui est donc indispensable de pouvoir déboguer son code lorsqu'une exception survient.
Les outils de Visual Studio 2010 pour Windows Phone 7 intègrent non seulement un émulateur (nécessaire pour permettre la réalisation d'applications tant que l'on ne dispose pas du périphérique adéquat) mais aussi un débogueur intégré directement dans Visual Studio :

```
private void listBox1_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    ListBoxItem listBoxItem = e.AddedItems[0] as ListBoxItem;
    if(listBoxItem != null)
    {
        string url = string.Format("/Page1.xaml?site={0}", listBoxItem.Tag);
        NavigationService.Navigate(new Uri(url, UriKind.Relative));
    }
}
```
url = "/Page1.xaml?site=http://www.programmez.com"

Il est possible de mettre des points d'arrêt, des conditions, de faire du pas à pas, etc. Les conditions et les méthodologies de développement sont donc entièrement les mêmes que pour du Silverlight « normal », ou du WindowsForms, de l'ASP.NET, etc.

Conclusion

Nous n'avons vu ici qu'une petite partie des possibilités de développement d'applications Silverlight pour Windows Phone 7. Cependant, vous avez à présent tous les éléments nécessaires pour vous lancer dans la réalisation de ce type d'applications et découvrir, par vous-même, les fonctionnalités telles que l' « ApplicationBar », les menus de type Panorama, etc.

■ Thomas **Lebrun**
Consultant / Formateur – MVP Client Application Development
http://blogs.developpeur.org/tom

Windows Phone 7 et le jeu avec XNA

Depuis quelques années maintenant, XNA a fait ses preuves en tant que surcouche à DirectX, facilitant le développement de jeux vidéo et d'applications multimédias pour le PC et la Xbox 360. Bâti autour de la plate-forme .Net, le framework offre une approche intuitive, et surtout simplifiée du cycle de développement d'un jeu vidéo, via des outils tels que le Content Pipeline pour la gestion des ressources ou encore la plateforme Live pour la mise en réseau des jeux.

Quel développeur n'a jamais rêvé de s'atteler facilement à la création d'un jeu vidéo ? XNA vient répondre à cette demande, pour les étudiants, les hobbyistes ou encore les studios indépendants.

Dans la précédente version du framework, les développeurs outre-Atlantique ont eu la chance de pouvoir s'essayer au développement mobile sur le Zune, puis à l'introduction du tactile multipoint et de l'accéléromètre avec le Zune HD. Aujourd'hui, pour notre plus grand bonheur (nous les Européens), le développement Zune a été mis de côté au profit du Windows Phone 7, un produit qui devrait être plus largement distribué. Si vous étiez déjà développeurs XNA pour Xbox 360 ou pour PC, vous allez donc pouvoir faire face à un public différent que celui présent sur le Windows Live Indie Games, mettre en place votre modèle économique et distribuer votre application sur le marketplace.

Les outils à posséder

Le développement de jeu vidéo pour mobile requiert la version 4 d'XNA. À l'heure actuelle, celle-ci est distribuée en CTP avec Visual Studio 2010 dans le kit de développement Windows Phone 7 : nous pourrons donc déployer nos applications dans un émulateur du téléphone. La grande nouveauté de cette mouture du framework, c'est l'accès aux spécificités du téléphone telles qu'elles sont disponibles lorsque vous écrivez une application Silverlight. Ainsi, vous pourrez bénéficier de l'écran tactile multipoints, du service de localisation, de l'accéléromètre, etc.

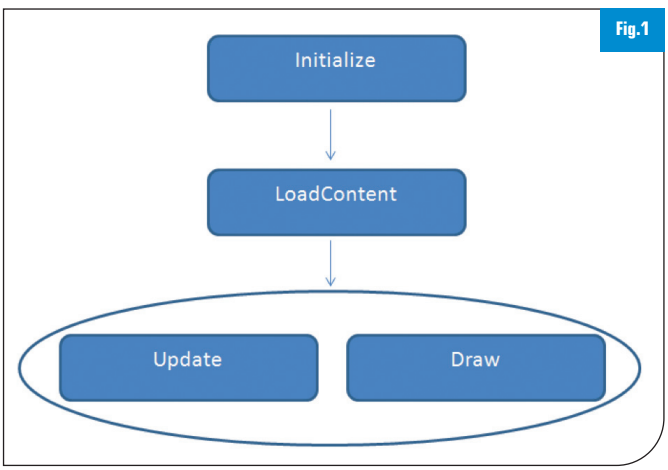


Fig.1

XNA, comment ça marche ?

Commençons par nous intéresser à XNA pour le développement d'un jeu Windows. Lorsque nous allons créer un nouveau projet, du code sera automatiquement généré. Dans ce code, on retrouvera une structure de programme qui est la suivante [Fig.1].

La méthode Initialize vous permettra de définir vos objets. La méthode LoadContent vous permettra de faire appel au mécanisme de chargement de données de XNA. Enfin, les méthodes Update et Draw sont automatiquement appelées à chaque trame. Dans la première vous placerez toute la logique de votre jeu (calculs de positions, interactions avec l'utilisateur) et dans la deuxième vous dessinerez vos objets. Il est important de noter que vous n'êtes pas obligé de respecter ce modèle, il est là uniquement pour vous guider…

L'intérêt de XNA est très simple : pour charger et afficher une image, vous n'avez besoin que de quelques lignes de codes.

```
Texture2D _texture;

protected override void LoadContent()
{
 spriteBatch = new SpriteBatch(GraphicsDevice);
 _texture = Content.Load<Texture2D>(@»mytexture»);
}

protected override void Draw(GameTime gameTime)
{
 GraphicsDevice.Clear(Color.CornflowerBlue);

 spriteBatch.Begin();
 spriteBatch.Draw(_texture, Vector2.Zero, Color.White);
 spriteBatch.End();
```

```
  base.Draw(gameTime);
}
```

On commence par créer un objet pour stocker notre image. On charge l'image en utilisant le mécanisme de gestion de contenu d'XNA. Et enfin, on l'affiche à l'écran. En ce qui concerne l'affichage, l'objet spriteBatch correspond à un shader (un programme exécuté par la carte graphique), auquel on passe la texture à dessiner, sa position et une teinte.
Ce qui est particulièrement intéressant dans XNA, c'est que ce modèle de chargement / affichage des données est reproductible pour un grand nombre de types de base livrés avec le framework (modèle 3d, pistes audio, etc.) et surtout, qu'il est extensible. Il vous est donc possible de créer des objets représentant une carte d'un jeu, de les charger et de les dessiner aussi facilement que cette texture. L'exemple ci-dessous illustre ce concept pour une vidéo.

```
Video _video;
VideoPlayer _player;

protected override void Initialize()
{
  _player = new VideoPlayer();

  base.Initialize();
}

protected override void LoadContent()
{
  spriteBatch = new SpriteBatch(GraphicsDevice);

  _video = Content.Load<Video>(@»video»);
}

protected override void Update(GameTime gameTime)
{
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back == Button
State.Pressed)
      this.Exit();

  if (_player.State == MediaState.Stopped)
      _player.Play(_video);

  base.Update(gameTime);
}
```

Migrons vers d'autres périphériques

Enfin, l'un des grands intérêts d'XNA, c'est sa portabilité. Comme pour les précédentes versions du framework, vous pouvez écrire un jeu pour le Windows Phone 7, puis facilement le convertir pour Windows ou Xbox 360 via un clic droit sur votre projet, et le choix de l'action intitulée « Create copy of Project for Windows Phone ». Un deuxième projet va apparaître dans la solution Visual Studio. Techniquement, retenez que les deux projets travaillent en fait sur les mêmes fichiers sources : les modifications que vous ferez sur le code de l'un seront automatiquement reportées sur le code de l'autre.

Lors du développement d'un jeu vidéo, il vous faudra faire attention aux spécificités des différentes plateformes : par exemple, il n'est pas question d'essayer de récupérer le statut de la souris ou du contrôleur de la Xbox 360 dans une application destinée au téléphone. Utilisez donc des directives préprocesseur pour garder la compatibilité dans votre code :

```
#if WINDOWS_PHONE
            Window.Title = «Windows Phone 7»;
#elif WINDOWS
            Window.Title = «Windows»;
#endif
```

Le développement pour Windows Phone 7

Parlons à présent des spécificités d'XNA 4.0 et relatives au Windows Phone 7. Notez que pour l'heure, l'émulateur livré avec les outils de développement ne supporte pas certaines de ces fonctions. Cependant, certains développeurs ont publié des outils tiers pour simuler ces fonctions. N'hésitez donc pas à fréquenter les sites communautaires et notamment le site officiel du framework : http://creators.xna.com/, où vous pourrez vous tenir au courant des dernières nouveautés.
En tant que développeur, vous devrez tout d'abord faire attention à la résolution de l'écran du téléphone. Les spécifications font référence à deux modes : 800 par 480, qui sera disponible au lancement du périphérique, et des écrans d'une résolution de 480 par 320 qui devraient faire leur apparition plus tard. Autre point intéressant : le framerate de vos jeux est par défaut bridé à 30 images par seconde car l'écran ne supporte pas un taux de rafraîchissement plus important.
Notez également, qu'à l'heure actuelle, sur l'émulateur, il n'existe pas de support pour des jeux en mode paysages mais que cela fera son apparition sur une prochaine version des outils. Pour reproduire ce comportement, vous devrez vous-même inverser les axes X et Y de vos jeux.
Le support de l'accéléromètre est apporté par l'assemblage Microsoft.Devices.Sensors. Le principe est simple, on initialise un objet de type AccelerometerSensor et on se branche sur un évènement qui sera levé dès que la valeur de l'une de ses composantes (X, Y ou Z) est modifiée.

```
AccelerometerSensor sensor = new AccelerometerSensor();
Vector3 sensorValues;

sensor.ReadingChanged += (s, e) =>
{
  sensorValues = new Vector3((float)e.Value.Value.X, (float)
e.Value.Value.Y, (float)e.Value.Value.Z);
};
```

Même si l'accéléromètre fait partie des spécifications minimales du téléphone, celui-ci n'est pas utilisable dans l'émulateur, ou tout simplement pas accessible depuis votre application ; dans ce cas, vous pouvez à tout moment consulter son état pour éviter des erreurs dans vos programmes.
Les spécifications minimales font également référence à un capteur de lumière et de présence. Le framework semble encore incomplet en ce qui concerne ces fonctions, mais dans le futur,

elles devraient être intégrables à vos jeux vidéo. En ce qui concerne le tactile multipoint, les spécifications matérielles du téléphone nous garantissent un minimum de quatre points de contact.
Cette fonctionnalité est axée autour des méthodes statiques de la classe TouchPanel. Nous pouvons par exemple commencer par récupérer l'état actuel du périphérique.

```
TouchPanelCapabilities touchPanelCapabilities = TouchPanel.
GetCapabilities();
bool isTouchPanelAvailable = touchPanelCapabilities.IsConnected;
```

XNA a pour habitude d'unifier au maximum les différentes plateformes. Ainsi, la procédure de récupération de l'état des différents points de contacts du téléphone est assez semblable à celle de récupération de l'état d'une manette de la Xbox 360 ou à celle du clavier du PC. Nous allons appeler la méthode GetState du TouchPanel, qui nous renverra un ensemble de TouchLocation.

```
TouchCollection touchCollection = TouchPanel.GetState();
```

La première chose que nous pouvons ensuite faire, c'est récupérer le type de point de contact : est-ce une simple pression ou est-ce un déplacement ? C'est le but de l'énumération TouchLocationState, accessible via la propriété State d'un objet de type TouchLocation.
Il est ensuite possible d'accéder à la position du point de contact via sa propriété Position qui nous renverra un jeu de coordonnées X et Y. Dans le cas d'un mouvement, les développeurs ont mis à notre disposition une méthode TryGetPreviousLocation. Il est alors possible de récupérer la précédente position et d'en tirer des informations, par exemple un calcul d'accélération.
Autre nouveauté de XNA 4.0, c'est le support de l'entrée audio via un microphone. Le tout s'axe autour d'une classe Microphone : une méthode Start, une méthode Stop et une méthode GetData nous permettant de récupérer le contenu du buffer au format défini par une propriété SampleRate.
En vrac, citons également le support de l'accélération 3D sur le téléphone (ce qui devrait vous laisser assez rêveur quant aux possibilités qui s'offrent à vous), et le support de la plateforme Live. Ce dernier point signifie qu'un joueur pourra se connecter à une plateforme de jeux unifiée pour la Xbox, Windows et maintenant le Windows Phone. Du côté des développeurs, il sera ainsi possible de récupérer des informations sur le joueur, tels que son gamertag ou son avatar.

Un premier jeu en pas à pas

Nous allons à présent réaliser un premier jeu exploitant les capacités du téléphone. Dans celui-ci, nous ferons apparaître des balles là où nous détecterons qu'il y a contact avec l'écran tactile. Nous utiliserons ensuite l'accéléromètre pour déplacer ces objets.
Commençons par créer une classe qui représentera l'entité balle. Celle-ci sera composée d'une texture (une image à l'écran) ainsi que d'une position.
Ajoutons également des méthodes pour le chargement de la texture, pour son affichage ainsi que pour les déplacements vers la gauche ou vers la droite. Dans ces deux dernières méthodes, avant de modifier la position des images, nous testons tout d'abord si elles ne sortent pas de l'écran. Notez d'ailleurs qu'avec XNA, le point d'origine des textures est placé en haut à gauche de celles-ci. Ce qui explique que dans la méthode GoRight, nous ayons à additionner la valeur de la composante X de la position de l'image à sa largeur pour vérifier sa position.

```
public class Ball
{
  private Texture2D _texture;

  public Vector2 Position { get; set; }

  public void LoadContent(ContentManager content)
  {
      _texture = content.Load<Texture2D>(@»ball»);
  }

  public void Draw(SpriteBatch spriteBatch)
  {
    spriteBatch.Draw(_texture, Position, Color.White);
  }

  public void GoLeft()
  {
    if (Position.X > 0)
      Position = new Vector2(Position.X - 1, Position.Y);
  }

  public void GoRight()
  {
    if (Position.X + _texture.Width < 480)
      Position = new Vector2(Position.X + 1, Position.Y);
```

Fig.2

Fig.3

```
    }
  }
```

Maintenant que cette classe est prête, il faut l'utiliser dans notre jeu. Pour cela, commençons par créer une collection de Ball.

```
public class Game1 : Microsoft.Xna.Framework.Game
{
  private List<Ball> _balls = new List<Ball>();

    // …
```

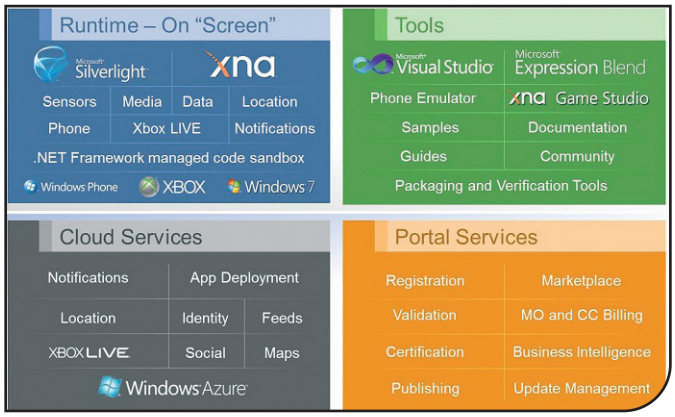
Dans la method Draw, nous pouvons dès maintenant parcourir la collection pour dessiner l'ensemble de ses éléments.

```
protected override void Draw(GameTime gameTime)
{
      GraphicsDevice.Clear(Color.CornflowerBlue);

    spriteBatch.Begin();

  foreach (Ball ball in _balls)
          ball.Draw(spriteBatch);

    spriteBatch.End();

  base.Draw(gameTime);
}
```

À présent, nous devons peupler la collection. Dans la méthode Update, nous allons devoir récupérer l'ensemble des points qui sont actuellement touchés par l'utilisateur. À partir de là, nous pouvons créer une nouvelle instance de Ball, lui assigner la position du point touché par l'utilisateur, charger sa texture et enfin l'ajouter à la collection.

```
protected override void Update(GameTime gameTime)
{
  // Allows the game to exit
  if (GamePad.GetState(PlayerIndex.One).Buttons.Back == Button
State.Pressed)
    this.Exit();

  TouchCollection touchCollection = TouchPanel.GetState();

  foreach (TouchLocation touchLocation in touchCollection)
  {
    Ball ball = new Ball { Position = touchLocation.Position };
        ball.LoadContent(Content);
      _balls.Add(ball);
  }

  base.Update(gameTime);
}
```

Il ne nous reste plus qu'à réagir à l'accéléromètre. Dans la méthode Initialize, nous allons créer une instance du capteur et via une expression lambda, nous récupérerons son statut dès qu'il sera mis à jour. Inclinez le périphérique sur la droite et sa composante X sera positive. À l'inverse, inclinez-le sur la gauche et la composante sera négative. Il faut donc appeler les méthodes GoLeft ou GoRight selon les valeurs de l'accéléromètre.




```
protected override void Initialize()
{
  base.Initialize();

  AccelerometerSensor sensor = new AccelerometerSensor();

  sensor.Start();

    sensor.ReadingChanged += (s, e) =>
    {
      if (e.Value.Value.X > 0)
              _balls.ForEach(b => b.GoRight());
                    else
      _balls.ForEach(b => b.GoLeft());
    };
}
```

Les possibilités du téléphone

Si vous vous demandez ce qui est techniquement réalisable avec XNA pour Windows Phone 7, voici deux captures d'écran en guise de mise en bouche [Fig.2 et3].

Enfin terminons par le modèle économique de XNA pour Windows Phone 7. Il n'y aurait pas de licence premium à acheter comme c'est le cas pour un développement dont la cible est la Xbox 360, mais plutôt des frais de publication à hauteur de 99$ par an pour 5 applications gratuites et un nombre illimité d'applications payantes. Cela dit, les étudiants seront ravis puisqu'ils seront exempts de ces frais.

XNA est donc un framework facile d'accès, qui met toute la puissance du Windows Phone 7 à portée des personnes ne s'étant jamais exercé au développement de jeux vidéo. La seule limite qui s'impose dans la réalisation d'un jeu pour mobile : **votre imagination**.

■ Léonard **LABAT**
Microsoft Student Partner
http://blogs.dotnet-france.com/leonardl

Un robot parleur sous Android

Parmi les nombreuses fonctionnalités du SDK d'Android il existe un service de synthèse vocale qui vous permet de faire dire à votre téléphone ce que vous voulez dans plusieurs langues. Les applications Android peuvent facilement tirer parti de cette API afin d'aider les personnes mal ou non voyantes, ou permettre de lire à voix haute un SMS qui arrive, lorsque vous conduisez votre voiture par exemple. Dans cet article nous allons nous intéresser à la mise en œuvre d'une petite application de robot parleur qui prononce dans la langue choisie un message saisi au clavier.

L'application est composée d'une activité principale proposant une IHM pour saisir le texte, et d'une activité secondaire accessible depuis un menu d'options, permettant de modifier les réglages de notre application. Les constantes de l'application sont centralisées dans l'interface Constants, tandis que les fonctions de synthèse vocale sont encapsulées dans la classe Tts. Des fichiers de ressources sont utilisés pour définir l'IHM de l'application (layout main.xml), le menu d'options (options_menu.xml) et les différentes options de réglages (settings.xml). Tous les textes de l'application sont externalisés dans values/strings.xml, c'est la ressource utilisée par défaut ; pour une version bilingue il suffit de créer une ressource values-fr/strings.xml avec les textes en français [Fig.1]. Manifeste de l'application :

```
<?xml version=»1.0» encoding=»utf-8»?>
<manifest xmlns:android=»http://schemas.android.com/apk/res/android» android:versionCode=»1» android:versionName=»1.0» package=»com.programmez.android.speechbot»>
 <application android:icon=»@drawable/icon» android:label=»@string/app_name»>
    <activity android:name=».App» android:label=»@string/app_name»>
      <intent-filter>
        <action android:name=»android.intent.action.MAIN»/>
        <category android:name=»android.intent.category.LAUNCHER»/>
      </intent-filter>
    </activity>
    <activity android:name=».Settings»/>
  </application>
  <uses-sdk android:minSdkVersion=»4» android:targetSdkVersion=»7»/>
</manifest>
```

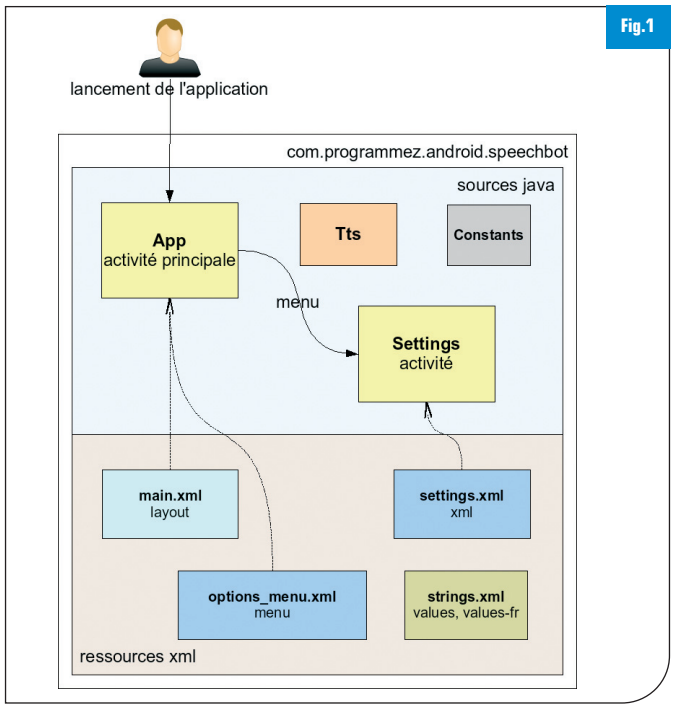


Fig.1

Fig.2

Activité principale et IHM

L'interface de l'application est constituée d'un titre, une invite à saisir, une zone d'édition et un bouton pour lancer la synthèse vocale. [Fig.2]. L'activité principale est prise en charge par la classe App, qui affiche le layout et gère les interactions avec l'utilisateur :

- La première fois que l'utilisateur lance l'application, un texte par défaut lui est proposé (« Bonjour !»)
- Chaque fois que le texte est modifié, il est sauvegardé et remplacera le texte par défaut pour les prochains lancements de l'application.
- Lorsque l'utilisateur clique sur le bouton « Parler », l'activité lance la synthèse vocale.
- Un menu d'options est accessible via la touche « menu » du téléphone, avec la possibilité d'accéder aux réglages ou de quitter l'application.

Source de App.java :

toutes les sources sont disponibles sur le site de Programmez !.

La méthode onCreate est invoquée à la création de l'activité lorsque l'application est lancée depuis le bureau; elle se contente de récupérer une instance du service TTS, de définir le layout puis de construire la zone de texte et le bouton.

Ces deux widgets sont associés à des écouteurs :

- pour la zone de texte : la méthode onTextChanged est invoquée à chaque modification de texte; c'est l'endroit idéal pour effectuer la sauvegarde du texte.
- pour le bouton : à chaque clic on lance la synthèse vocale, qui est déléguée à la méthode speech,

Les méthodes onCreateOptionsMenu et onMenuItemSelected prennent en charge le menu d'options :

- onCreateOptionsMenu crée le menu : on a choisi ici de définir le menu dans un fichier XML plutôt qu'en java ; d'où l'utilisation de MenuInflater.
- onMenuItemSelected gère la sélection parmi les deux options :
  - option « Réglages » : lancement d'une activité fille (la deuxième activité du projet) via une intention.
  - option « Quitter » : on quitte l'application en terminant simplement l'activité.

Pour identifier chaque option, on utilise la fameuse classe « magique » R générée à la volée à partir des fichiers de ressources. Le lancement de l'activité fille (Settings) s'effectue avec startActivityForResult car on souhaite être informé de sa fin (retour à l'activité principale une fois les réglages terminés).
L'événement de retour se matérialise par l'invocation de la méthode onActivityResult, quelle que soit la provenance.
S'il était utile de connaître la provenance, il suffirait de vérifier la valeur du paramètre requestCode (ici notre constante MENU_SETTINGS_REQUEST_CODE); on se contente ici de rafraîchir la zone de texte afin de gérer un éventuel changement effectué par l'activité fille. Enfin, on redéfinit la méthode onDestroy pour libérer proprement les ressources du service TTS avant de détruire l'activité.

Constantes du projet :

```
package com.programmez.android.speechbot;
public interface Constants {
  String APP_NAME = «SpeechBot»;
  String LOG_TAG = APP_NAME;
  int MENU_SETTINGS_REQUEST_CODE = 1;
  int TTS_CHECK_REQUEST_CODE = 2;
  String PREF_KEY_CHECK_TTS = «PREF_KEY_CHECK_TTS»;
  String PREF_KEY_EDIT_TEXT = «PREF_KEY_EDIT_TEXT»;
  String PREF_KEY_LANGUAGE = «PREF_KEY_LANGUAGE»;
  String PREF_KEY_RESET = «PREF_KEY_RESET»;
  String PREF_KEY_SPEECH_RATE = «PREF_KEY_SPEECH_RATE»;
  int TTS_QUEUE_FLUSH = 0;
  int TTS_QUEUE_ADD = 1;
}
```

Le menu d'options de l'application est défini dans la ressource res/menu/options_menu.xml :

```
<?xml version=»1.0» encoding=»utf-8»?>
<menu xmlns:android=»http://schemas.android.com/apk/res/android»>
  <item android:title=»@string/settings» android:icon=»@android:drawable/ic_menu_preferences» android:id=»@+id/options_menu_settings»/>
  <item android:title=»@string/quit» android:id=»@+id/options_menu_quit» android:icon=»@android:drawable/ic_menu_close_clear_cancel»/>
</menu>
```

[Fig.3]

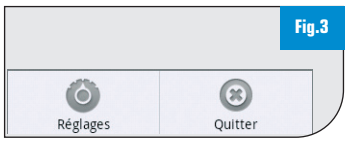


Fig.3

Le menu d'options s'affiche en appuyant sur la touche « Menu » du téléphone.

Activité secondaire : réglages de l'application

L'activité secondaire est implémentée par la classe Settings.
Étant donné sa vocation, elle étend la classe PreferenceActivity pour bénéficier de comportements standard de gestion de préférences; nul besoin de définir une IHM, tout est automatique.
L'ensemble des réglages (préférences) est défini dans un fichier de ressources XML, avec :

- Le choix de la langue
- Le choix de la vitesse de diction
- La possibilité de vérifier la disponibilité du service TTS
- La possibilité de réinitialiser tous les réglages

Source de la ressource res/xml/settings.xml :

```
<?xml version=»1.0» encoding=»utf-8»?>
<PreferenceScreen xmlns:android=»http://schemas.android.com/apk/res/android»>
  <ListPreference android:key=»PREF_KEY_LANGUAGE» android:title=»@string/language» android:summary=»@string/language_summary» android:persistent=»true»/>
  <ListPreference android:summary=»@string/speech_rate_summary» android:title=»@string/speech_rate» android:key=»PREF_KEY_SPEECH_RATE» android:entryValues=»@array/speech_rate_values» android:entries=»@array/speech_rates» android:defaultValue=»100»/>
  <Preference android:title=»@string/check_tts» android:key=»PREF_KEY_CHECK_TTS» android:summary=»@string/check_tts_summary»/>
  <Preference android:key=»PREF_KEY_RESET» android:summary=»@string/reset_summary» android:title=»@string/reset»/>
</PreferenceScreen>
```

Chaque préférence est associée à une clef qui sera utilisée pour la persister et qui servira de donnée discriminante aux traitements java.

Source de Settings.java, explication :
Les trois méthodes statiques sont de simples helpers, qui auraient pu classiquement appartenir à une classe « Util » (on économise une classe). La méthode onCreate récupère une instance du service TTS, ajoute les préférences depuis le fichier de ressources, et construit la liste des langues disponibles. Les deux dernières lignes associent des écouteurs aux options simplement cliquables : la vérification du service TTS et la réinitialisation. La méthode buildAvailableLanguages définit la liste des langues qui sera proposée, avec comme premier choix la langue par défaut. C'est la classe Tts qui fournit les langues effectivement supportées.
Dans la méthode onPreferenceClick, on effectue le traitement correspondant à la clé de l'option sélectionnée :

- Vérification du service TTS : on invoque la méthode checkTtsData qui lance une activité de vérification des données propre au service Tts, on récupère ensuite le code résultat dans la méthode onActivityResult que l'on transmet à nouveau au service Tts pour déléguer la vérification. Si tout est opérationnel on affiche un message.
- Réinitialisation : on récupère les préférences de l'application, on efface tout puis on termine l'activité (les préférences ne sont plus à jour).

[Fig.4, 5, 6]
L'activité PreferenceActivity du SDK d'Android présente des avantages certains de simplicité et rapidité de mise en œuvre ; elle

répond à la majorité des besoins mais fournit un look & feel basique, et ne conviendra donc pas toujours à des besoins plus exotiques.

La synthèse vocale : service TTS (Text-to-speech)

Pour effectuer la synthèse vocale, le SDK d'Android fournit en standard un module de Text-to-speech à partir de la version 1.6 (Donut). Notre classe Tts sert de façade dans le projet pour accéder à ce service technique du SDK. La classe Tts implémente le pattern singleton (methode getInstance), pour faciliter le partage du service dans les différentes activités quel que soit leur cycle de vie. La constitution de la liste des langues supportées est assurée par la méthode obtainAvailableLocales qui se contente de renvoyer « en dur » deux langues. Pour obtenir dynamiquement la liste des langues réellement supportées on pourrait faire évoluer cette méthode en respectant le principe suivant :

- Récupération des langues du système Android,
- Pour chaque Locale, vérification de sa disponibilité au sein du module TTS (méthode isLanguageAvailable),
- Tri et nettoyage des doublons dans la liste.

La méthode checkTtsData permet de vérifier les données du module TTS en lançant une activité dédiée (ACTION_CHECK_TTS_DATA).

La méthode onCheckTtsDataResult interprète le résultat de l'activité et lance l'installation du module TTS si le test a échoué. Les autres méthodes jouent leur rôle de façade et utilisent l'API du module TTS pour définir la langue (setLanguage), définir la vitesse (setSpeechRate), libérer les ressources (shutdown), et parler (speak). La méthode speak peut être invoquée pour ajouter un texte à la file d'attente (mode QUEUE_ADD) ou le dire immédiatement et vider la file (mode QUEUE_FLUSH). En cas d'erreur du module TTS, une exception spécifique est levée :

```
package com.programmez.android.speechbot.tts;

public class TtsNotInitializedException extends Exception {
  private static final long serialVersionUID = -8307341316965
013306L;
}
```

Problématique de fragmentation : TTS avec Cupcake ?

Le service mis en œuvre s'appuie sur le module intégré à Android depuis la v1.6 (Donut). Tous les androphones récents propulsés par Donut ou Eclair (v2.0-2.1) seront donc compatibles.

Mais ceux qui sont propulsés par une version antérieure comme Cupcake (v1.5), encore bien présente sur le marché malheureusement, poseront problème (Samsung Galaxy par exemple).

[Fig.7 et 8]

Comment faire fonctionner l'application avec Cupcake ?

Il se trouve qu'un module tiers existe via le projet Google « eyes-free » : http://code.google.com/p/eyes-free/

En fait, le module intégré au SDK est postérieur à ce projet qui continue d'évoluer.

Les livrables se présentent sous deux formes :

- Une application pour les utilisateurs : « Text-To-Speech Extended »
- Une librairie (.jar) pour les développeurs : http://eyesfree.googlecode.com/files/TTS_library_stub_1.7_market.jar

Fig.9

[Fig.9]

L'API fournie par la librairie est relativement similaire au module standard Donut, au nom de package près et à quelques petites différences non bloquantes. En revanche, le rendu vocal n'est pas du tout le même, c'est un homme qui parle au lieu d'une femme et la voix est beaucoup plus « robotisée »… Les réglages intrinsèques du module sont accessibles via l'application Text-To-Speech Extended, au lieu du panneau de configuration standard.

Intégration de Text-To-Speech Extended

L'intégration de ce module à notre application est un moindre mal puisque nous bénéficions d'une façade; il reste néanmoins à savoir si l'application doit régresser en 1.5 ou rester au-dessus, et savoir déterminer dynamiquement la version du téléphone hôte pour choisir le bon module.

Le passage en v1.5 du projet dans Eclipse génère des erreurs liées à la perte des API supérieures.

Une première solution consiste à utiliser la réflexion pour instancier dynami-

Fig.7

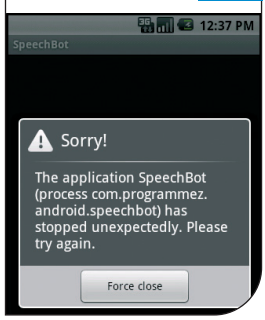


Fig.8

```
AndroidRuntime    Uncaught handler: thread main exiting due to uncaught exception
AndroidRuntime    java.lang.VerifyError: com.programmez.android.speechbot.tts.Tts
AndroidRuntime        at com.programmez.android.speechbot.SpeechBot.onCreate(SpeechBot.java:33)
AndroidRuntime        at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1123)
AndroidRuntime        at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2231)
AndroidRuntime        at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2284)
AndroidRuntime        at android.app.ActivityThread.access$1800(ActivityThread.java:112)
AndroidRuntime        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1692)
AndroidRuntime        at android.os.Handler.dispatchMessage(Handler.java:99)
```

Fig.4

Fig.5

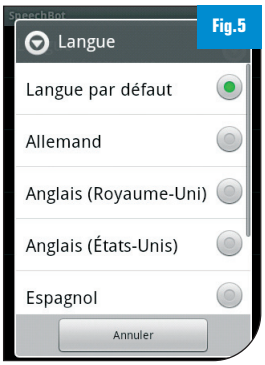


Fig.6

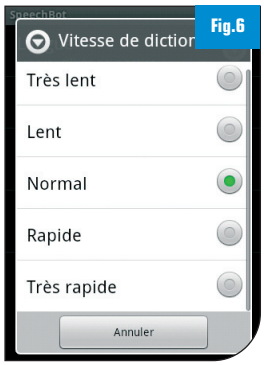
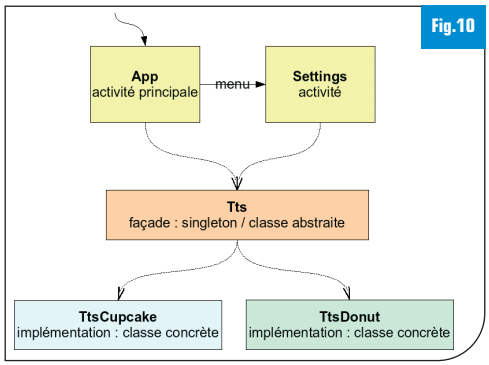


Fig.10



quement les classes nécessaires, invoquer leurs méthodes, accéder aux champs.
C'est techniquement tout à fait envisageable, mais le code source résultant y perdrait en qualité (compilateur impuissant), et serait pénible à maintenir, c'est un euphémisme.
Une autre solution consiste à laisser le projet en version haute (v2.1 par exemple), et changer dans le manifeste l'attribut minSdkVersion (3 pour Cupcake).

```
  <uses-sdk android:minSdkVersion=»3» android:targetSdkVersion=
»7» />
```

Il faut ensuite modifier la classe Tts pour la dissocier de l'implémentation effective; pour cela une classe abstraite est tout à fait appropriée.[Fig.10]. Voici les modifications à appliquer :

```
package com.programmez.android.speechbot.tts;
import java.util.*;
import android.app.Activity;
import android.content.Context;
import com.programmez.android.speechbot.App;
public abstract class Tts {
  protected static Tts instance;
  public static Tts getInstance(final Context context, final
boolean initialize) {
    if (instance == null || initialize) {
      try {
        instance = new TtsDonut(context);
      } catch (final VerifyError e) {
        instance = new TtsCupcake(context);
      }
    }
   App.log(«tts instance : « + instance.getClass().getSimple
Name());
    return instance;
  }
  protected boolean initialized;
  protected final Context context;
  protected Tts(final Context context) {
    this.context = context;
    initTtsImpl();
  }
  public abstract void checkTtsData(final Activity activity,
 int requestCode);
 public abstract boolean onCheckTtsDataResult(final Activity
 activity,
      final int resultCode);
 public void setLanguage(final String language)
      throws TtsNotInitializedException {
    checkTtsImpl();
    setLanguageImpl(language == null || «».equals(language)
 ? Locale
        .getDefault() : new Locale(language));
 }
 public abstract void setSpeechRate(final float speechRate);
 public void shutdown() {
    App.log(«tts shutdown»);
    shutdownTtsImpl();
    initialized = false;
    instance = null;
  }
  public void speak(final String text, final int queueMode)
      throws TtsNotInitializedException {
    checkTtsImpl();
    speakImpl(text, queueMode);
  }
  protected void checkTtsImpl() throws TtsNotInitialized
Exception {
    if (getTtsImpl() == null || !initialized)
      throw new TtsNotInitializedException();
  }
  protected abstract Object getTtsImpl();
  protected abstract void initTtsImpl();
  protected abstract Map<String, Set<Locale>> obtainAllTts
SupportedLocales();
  protected abstract void setLanguageImpl(Locale locale);
  protected abstract void shutdownTtsImpl();
  protected abstract void speakImpl(String text, int queue
Mode);
}
```

A noter dans la méthode getInstance : l'utilisation de l'exception VerifyError pour choisir l'implémentation en fonction de la version d'Android. Deux autres classes concrètes hériteront de la classe Tts et prendront en charge les deux différents modules implémentés en fonction de la version d'Android. Le contrat du module TTS vu par l'application est fixé par notre classe abstraite : le découplage est bien assuré. La classe TtsDonut est équivalente à notre ancienne implémentation, c'est quasiment un déplacement de code. La classe TtsCupCake implémente les fonctionnalités de synthèse vocale avec le module tiers Text-To-Speech.
Évidemment, les API utilisées apparaissent comme dépréciées dans le source, mais dans cette classe particulière on s'autorise exceptionnellement l'utilisation de l'annotation SuppressWarnings. On retrouve des situations analogues qui nécessitent l'emploi de cette technique pour gérer proprement les différentes versions d'Android : gestion des contacts, lecture de SMS …

Conclusion

La façade Tts peut facilement évoluer pour proposer plus de possibilités, par exemple intégrer vos propres samples vocaux ou encore choisir le moteur de synthèse. Ce système pourra très simplement être intégré à n'importe quelle application Android pour apporter un plus aux utilisateurs. En outre, la façade permet de gérer correctement la problématique de fragmentation en s'appuyant sur un service de synthèse vocale tiers (eyes-free) dans le cas d'un androphone Cupcake qui n'intègre pas l'API TextToSpeech standard de Donut.
Pour les plus courageux, une application Android de traduction simultanée pourrait être imaginée moyennant l'ajout de fonctions de reconnaissances vocales et l'intégration d'une API du type Google Traduction ;-)

■ Olivier **Penhoat**
Consultant & Formateur
Valtech Training

egilia®

LEARNING

LE SPÉCIALISTE DE LA
FORMATION CERTIFIANTE
EN INFORMATIQUE
ET MANAGEMENT

Faire de vos succès
notre réussite

www.egilia.com

CONTACTEZ NOS CONSEILLERS FORMATION

N° National 0 800 800 900

APPEL GRATUIT DEPUIS UN POSTE FIXE

ANVERS . LIEGE . PARIS . LYON . LILLE . AIX-EN-PROVENCE .
STRASBOURG . RENNES . BRUXELLES
TOULOUSE . BORDEAUX . GENEVE . LAUSANNE . ZURICH .

Le « Service Bus » : au-delà des frontières du système d'information

Quelles que soient les plateformes qui les hébergent, toutes les applications des systèmes d'information des entreprises ont besoin de communiquer entre elles, de consommer des services externes ou d'exposer des fonctionnalités vers des tiers.

Les architectures orientées service (SOA) ont introduit au sein des entreprises la notion de bus d'entreprise appelé ESB (Enterprise Service Bus) qui permet d'apporter notamment de l'agilité au sein du système d'information. Très vite, les entreprises ont exprimé le besoin d'étendre ces architectures au-delà des frontières de l'entreprise afin de communiquer avec leurs partenaires, clients et filiales. Un enjeu phare auquel Microsoft répond à travers l'offre « Service Bus ».

Une connexion sécurisée et évolutive

Composant de Windows Azure AppFabric, le « Service Bus » fournit l'infrastructure équivalente à un ESB mais à l'échelle d'Internet [Fig.1].

Service Registry

Le « Service Bus » est un bus logiciel sur lequel des points de terminaison (endpoints) permettent de connecter soit des services, soit des clients. Chaque point de connexion est caractérisé par une URI qui l'identifie de manière unique dans Azure. Le « service Bus Naming System » définit le modèle de cette URI de la façon suivante : (tableau ci-dessous).

[schéma]://[espace de nom du service].servicebus.windows.net/[nom1]/[nom2]/...

[schéma]	Désigne les trois schémas d'URIs supportés par le « *Service Bus* » : ▪ "http" et "https" pour les points de connexion basés sur http. ▪ "sb " pour tous les autres points de terminaison basés sut TCP.
[espace de nom du service]	Une appellation unique qui identifie un projet au sein du « *Service Bus* »
[nom1]/[nom2]...	Les noms des services contenus dans le projet, organisés de manière hiérarchique.

Ainsi, le « Service Registry » permet de publier et de découvrir toutes les références aux points de terminaison au sein de la solution exposée. Les consommateurs (clients) peuvent donc se connecter au point de connexion relatif à l'espace de nom du service et récupérer un "Atom Feed" racine qui décrit un premier niveau de noms de services. Chaque nom de service donne accès à un deuxième niveau hiérarchique à travers un autre "Atom Feed" et ainsi de suite jusqu'à épuiser tous les noms de la solution.

Application Messaging Patterns

L'application "Messaging" est au cœur des fonctionnalités du « Service Bus » car elle fournit plusieurs modèles avancés d'envoi de messages et de connectivité entre les applications. Elle permet donc de s'affranchir des contraintes de la couche de transport en se chargeant de transmettre le message au bon destinataire.

Le « relay service»

Ce service offre la possibilité de construire des solutions capables de communiquer même en présence de pare-feu et de dispositifs de NAT. Le principe est simple : le service hébergé en local se connecte au « Relay Service » à travers un port de sortie et crée une 'socket' de communication bidirectionnelle rattachée à une adresse particulière qu'on nommera 'rendezvous address'. Le client peut ainsi communiquer avec le service en envoyant des messages à cette adresse, le « Relay Service » se chargera par la suite de relayer ces messages au service cible en utilisant la 'socket' déjà mise en place via un mécanisme appelé le forwarder. L'intérêt majeur de cette solution est qu'elle ne nécessite aucun port entrant ouvert sur le Pare-feu, ce qui optimise la sécurité lors de l'accès au service. Le Relay Service joue d'ailleurs dans cette configuration le rôle d'une DMZ [Fig.2]. L'utilisation du "Relay Service" permet de profiter des différents patterns d'envoi de messages : le mode unidirectionnel,

Fig.1

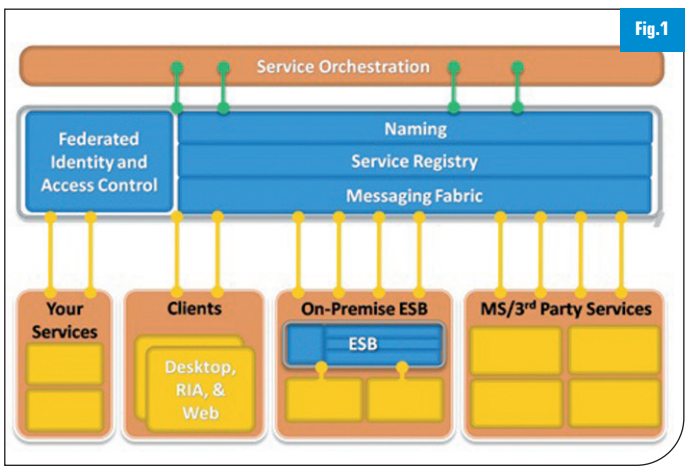
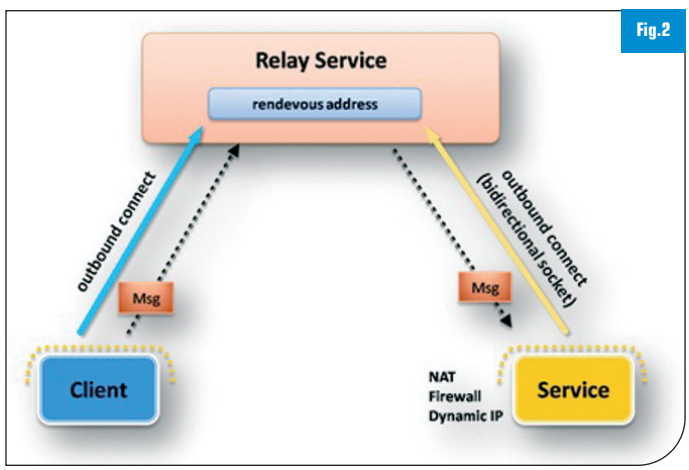


Fig.2



le mode requête/réponse, le mode pair-à-pair, le mode 'publish/subscribe' pour le multicast et le mode 'Message Buffer' pour les communications asynchrones. Le « Service Bus » offre également la possibilité d'établir une connexion directe entre les clients et les services afin d'améliorer les performances et le débit. Dans cette approche, les clients et les services communiquent dans un premier temps à travers le "Relay Service". Celui-ci va tenter, grâce à un algorithme de prédiction de port, de déterminer quels seront les ports ouverts dans leurs Pare-feu respectifs. Si l'information est correcte, la connexion directe entre les clients et les services aboutira, sinon ils se contenteront d'une communication relayée par le 'Relay Service". Il ne s'agit en aucun cas de trouver une faille de sécurité mais simplement d'établir une communication directe entre deux partenaires si les règles de sécurité en vigueur le permettent.

Le Service est une extension du modèle WCF. Il offre une correspondance avec les principaux bindings WCF. Par exemple, le WebHttpBinding devient le WebHttpRelayBinding.

Pattern : Publish / Subscribe

Un des patterns d'intégration d'entreprise fréquemment utilisé est le publish/subscribe, c'est-à-dire permettre de router le message vers plusieurs destinataires en même temps, on parle également de multicast. Ce type de pattern peut être utile pour envoyer par exemple une demande de devis à un ensemble de fournisseurs. Avec le service bus, il suffit d'utiliser le binding NetEventRelayBinding pour mettre en place ce mécanisme.

Pattern : Communication asynchrone - Message Buffer

Dans ce modèle, un client (producteur de message) envoie des messages au « Message Buffer », qui n'est autre qu'une structure de données avec une politique FIFO (First In First Out). Le service a la possibilité de définir la durée de vie du buffer à travers le « Message Buffer Policy » ainsi que la taille du message buffer par exemple. Ensuite, de manière asynchrone, le Message Consumer récupère le message par http GET puis le supprime du message buffer via http DELETE. Le « compagnon » indispensable du « Service Bus » est « l'Access Control Service ». Ce dernier traite les processus d'authentification et d'autorisation à l'accès du « Service Bus » afin d'apporter la sécurisation des échanges.

Quelle interopérabilité avec les technologies autres que celles de Microsoft ?

Il est évident que toute la panoplie des services offerts par Windows Azure, dont le « Service Bus », fonctionne parfaitement avec les technologies Microsoft. Mais qu'en est-il des technologies tierces : PHP, Java, ... ?

Un soin particulier a été apporté par Microsoft sur l'interopérabilité et l'ouverture de sa plateforme Cloud Computing et notamment la partie AppFabric. Cette interopérabilité est présente à la fois pour exposer ses services sur le cloud mais également pour les consommer. On peut d'ailleurs imaginer des scénarios faisant appel au Service Bus d'AppFabric entre deux applications non Microsoft, l'une développée en Java et la seconde en Ruby. De plus, les entreprises vont de plus en plus consommer des services tiers provenant de fournisseurs ayant investi sur des plateformes Cloud Computing différentes, par exemple Amazon ou Google pour ne citer qu'eux.

En tant que plateforme ouverte, Microsoft offre le choix aux développeurs. Ils peuvent utiliser différents langages comme .Net, Php, Ruby, Python ou Java et différents outils de développement comme Visual Studio et Eclipse pour développer et consommer les services à partir d'applications à demeure ou dans les nuages.

Quel avenir pour le « Service Bus » ?

Le « Service Bus » accompagné de l'Access Control Service est un des éléments différenciant de la stratégie de Microsoft sur le Cloud Computing. Cette brique logicielle va permettre de faciliter les échanges entre solutions hébergées dans le cloud et les solutions à demeure. Alors que la SOA avait brisé l'approche silo des applications au sein d'une entreprise, le « Service Bus » va offrir la possibilité de briser les frontières du système d'information de manière sécurisée. Une fois ces services déployés dans le « Service Bus », les entreprises devront pouvoir superviser, mesurer l'utilisation des services afin de pouvoir notamment en refacturer l'utilisation en fonction de leur usage. Ces derniers services sont, de notre point de vue, ce qui est attendu dans les prochaines versions d'Azure.

■ Michel **Hubert**
Manager chez Logica Business Consulting en charge de l'offre Azure

■ Safae **Rhayour**
Consultante chez Logica Business Consulting

Stockage des données dans Windows Azure

La plateforme Windows Azure inclut différents types de stockage pour conserver nos données dans le Cloud. Nous pouvons utiliser ces différents espaces de stockage pour nos applications exécutées dans le cloud (Windows Azure Compute) ou nos applications « on-premise » (sur site) qui auraient besoin de déporter partiellement ou totalement la conservation des données dans le cloud (pour permettre une plus grande disponibilité et extensibilité du stockage). Parcourons ensemble ce que nous propose Microsoft au travers de la plateforme Azure.

La plateforme Azure propose deux services dédiés :

- Le Windows Azure Storage, mettant à disposition 3 types de stockage : les Blobs, les Tables et les Queues
- SQL Azure, qui comme son nom l'indique, met à disposition un véritable SGBD dans le Cloud

Nous retrouvons aussi la possibilité de stocker des données dans le « Local Storage » propre au service de Compute de la brique Windows Azure. Examinons ces différentes solutions.

Le Local Storage

Cet espace de stockage est propre au service de Compute, car il permet de stocker sur le disque de la machine virtuelle dans laquelle s'exécute votre application (web ou worker role). Cet espace est propre à une instance de votre service dans Windows Azure et n'est donc pas partagé entre les différentes instances de votre application, car rappelez-vous qu'une instance dans Windows Azure équivaut à une machine virtuelle. C'est pour cela qu'on parle de stockage « Local ». Physiquement, les données sont stockées sur l'espace disque de la machine virtuelle hébergeant votre rôle Azure.

Les données sont accessibles en utilisant les opérations standard des streams en .NET (Stream, File, Directory, etc.). Aucun accès réseau n'est requis car toutes les données sont en local. Pour créer son Local Storage, il faut le définir dans le fichier de définition du service (ServiceDefinition.csdef). La taille minium pour un « Local Storage » est de 1MB et son maximum est dépendant du type de l'instance sélectionné entre « Small », « Medium », « Large » et « Extra large », proposant respectivement un total de 225 GB, 490 GB, 1000 GB et 2040 GB. (Bien entendu la facturation ne sera pas la même suivant le type d'instance choisie). Pour l'utiliser il suffira de récupérer le chemin local du Local Storage via le RoleManager en appelant la méthode « GetLocalResource » qui prendra en paramètre le nom du LocalStorage déclaré dans le fichier de description du service (ServiceDefinition.cscfg). Libre à nous d'y stocker ce que l'on veut dans cet espace de stockage ! Mais attention, toutes ces données sont locales au rôle et ne sont pas accessibles par d'autres rôles ou instances. De plus, vous perdez le côté « load-balancing », « failover » (en cas de panne) et extensible (au mieux, limité à 2TB pour l'Extra large Instance).

Windows Azure Storage

La plateforme Windows Azure nous propose un service de « Storage » permettant de créer des « Storage Accounts » qui comprennent trois services de stockage :

- Les Azures Queues : un service de queues (de type FIFO – First In First Out)
- Les Azure Tables : des tables d'entité de type clé/valeur
- Les Azure Blob : un espace de stockage nommé,

Le « Storage Account » est totalement indépendant du service de Compute. D'ailleurs, il n'y a pas de lien direct entre vos comptes de Compute (« Hosted Services ») et les comptes de stockage (« Storage Account ») bien que vous puissiez choisir d'héberger vos « Hosted Service » et « Stora-

ge Account » dans la même unité géographique (afin d'optimiser les accès réseau entre vos services et données). Les Storage Account sont limités à 100 TB sachant qu'un abonnement Azure peut créer jusqu'à 5 « Storage Accounts ».

Les trois services du « Storage Account » sont accessibles et manipulables au travers des protocoles HTTP/HTTPS en REST. On peut donc se servir du Storage Account depuis n'importe quel service qu'il soit « In the Cloud » (Azure ou autre) ou « On Premise » et quelle que soit la technologie (.NET ou autre, du moment que celle-ci sache utiliser le protocole http). Vous trouverez à ce sujet des projets comme PHP Azure (http://phpazure.codeplex.com/) ou Windows Azure 4J (http://www.windowsazure4j.org/) proposant respectivement des librairies PHP et Java pour interagir facilement avec les services de blobs, tables et queues. En .NET, Microsoft nous met à disposition l'assembly « Microsoft.WindowsAzure.StorageClient » dans le SDK de Windows Azure. Cette librairie nous permettra d'interagir très facilement avec notre Storage Account. « Azure Table » comme son nom l'indique est un service de « Table » à ne pas confondre avec une base de données relationnelle comme nous avons l'habitude avec des produits de type SQL Server (et son équivalent « in the cloud » SQL Azure) bien que des ressemblances existent. Les Azure Tables sont de simples tables d'entités de type clé/valeur. Chaque table comporte une collection d'entités, chacune d'elle pouvant contenir jusqu'à 255 propriétés (d'un maximum de 64KB pour les String ou binaires). Ces tables sont exposées en http grâce à la brique ADO.NET Data Services, anciennement appelée Astoria. Les entités sont réparties dans des partitions (que vous définissez) permettant à Microsoft de garantir le load-balancing vers nos données. Aucun schéma n'est défini sur vos tables, vous pouvez donc stocker deux entités de type différent dans la même table.

Les « Azure Blobs » proposent un stockage de fichier nommé dans lequel on créera des Containers (équivalent d'un dossier) qui contiendront une collection de blobs (équivalent d'un fichier). L'interface REST nous proposera des méthodes permettant d'envoyer des données dans un blob, de les récupérer, supprimer, copier, etc. Chaque blob et container peut être associé à des métadonnées (paires de clé/valeur) permettant ensuite de requêter vos blobs en fonction de ses propriétés. On y trouve deux types de Blob, le plus courant les « BlockBlob » où chaque blob est une séquence de block permettant de stocker jusqu'à 200 GB par blob et les « PageBlob » permettant de paginer le blob en page afin de proposer un accès de type « random » et permettant de stocker jusqu'à 1 TB par blob.

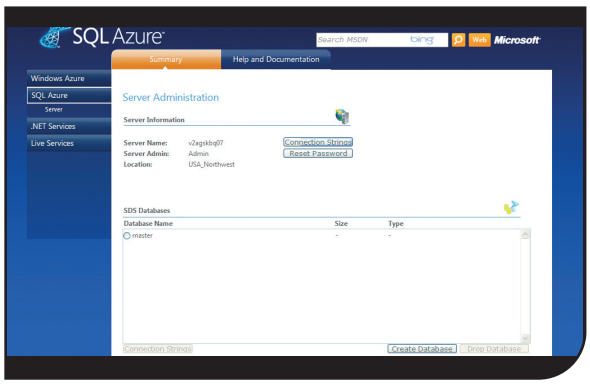
Depuis un web ou worker role sur le service de Compute, nous pouvons aussi utiliser les « Azure Drives » permettant de « monter » une partition NTFS. Cette partition n'est ni plus ni moins qu'un disque virtuel (VHD) stocké sur le service de Blob. Cela nous permet d'utiliser un espace de stockage NTFS persistant et disponible depuis un service Windows Azure.

Pour finir les « Azure Queues » mettent à disposition un service de queue de type FIFO (1er entré 1er sorti) dans lequel nous allons pouvoir envoyer des messages qui contiendront une valeur en String ou un Stream. Il n'y a pas de limite dans le nombre de messages dans la queue, mais chaque message ne doit pas excéder une taille de 8 KB. Il sera possible de créer et supprimer autant de queues que l'on désire (identifié par un nom) et chaque queue permettra d'envoyer des messages (put), de les récupérer (get) et de les supprimer (delete).

SQL Azure

Afin de compléter l'offre de stockage sur la plateforme Windows Azure, Microsoft met à disposition la brique « SQL Azure » nous offrant un véritable système de base de données relationnel avec tous les avantages du Cloud Computing. SQL Azure n'est ni plus ni moins qu'une version spéciale de « MS SQL Server » pour le cloud.

SQL Azure propose deux éditions : « Web » limitée à 1 GB, et « Business » limitée à 10 GB (et bientôt 50 GB). Chaque abonnement peut créer une ou plusieurs bases de données dans la limite de taille de l'édition choisie. Chaque base de données contient les objets standard SQL (table, vue, index, utilisateur, procédures stockées, triggers, contraintes, ..).



Console SQL Azure.

Ces bases de données sont physiquement stockées sur plusieurs serveurs SQL, nommés *réplicas,* afin d'assurer la disponibilité et montée en charge de vos bases. En cas de panne d'un des *réplicas* hébergeant vos bases, vos données restent toujours disponibles grâce aux autres *réplicas.*

Les bases sont accessibles en TDS (protocole standard de MS SQL) permettant d'utiliser SQL Azure comme un serveur local, SQL Server limitant ainsi les efforts en cas de migration.

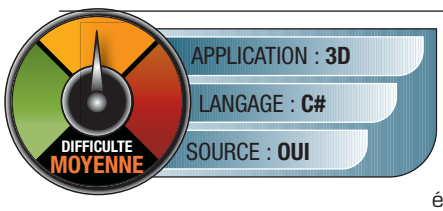
Les connexions entrantes sont sécurisées (SSL) et centralisées sur une passerelle qui gère la sécurité des accès (authentification, autorisation, règle du firewall) et répartit les requêtes sur les *réplicas* SQL contenant vos bases. L'administration physique est donc complètement transparente en nous proposant une réplication de vos bases de manière automatique, un fail-over transparent en cas de panne et un load-balancing des requêtes pour garantir le SDL. Le rôle du DBA sera plus axé sur l'administration logique en mettant le focus sur la création et le management des schémas, l'optimisation des requêtes et la sécurité des accès (login, user, role, ...).

■ Sébastien **Warin**
Technical Lead- R&I Wygwam - Usilink
http://sebastien.warin.fr

DirectX 11 : DirectCompute

2e partie

Faisant suite à l'article de découverte de DirectX11, nous allons nous pencher sur une des fonctionnalités majeures de cette version : Les Compute Shaders. Ce mécanisme va nous ouvrir les portes d'une technologie très puissante : le GPGPU (pour Global Purpose GPU). C'est par ce biais que nous allons pouvoir accéder à la force de calcul des GPU et à leur grande capacité de traitements parallèles. Il est à noter que nous utiliserons à nouveau SlimDX pour le wrapper sur DirectX.



Le GPGPU tire sa source de l'arrivée des premières cartes graphiques programmables via des shaders. Par la suite, des éditeurs comme Nvidia ont su proposer des outils de programmation permettant de dérouter la puissance de génération d'images vers des services plus métiers. C'est ainsi qu'est apparu CUDA ou plus récemment OpenCL.

Microsoft a donc profité de la sortie de DirectX11 pour doter son API d'une solution pour faire du GPGPU. Cette solution se nomme DirectCompute et s'appuie sur l'apparition d'une nouvelle famille de shaders : les Compute Shaders.

Ces derniers ont pour vocation de faire du traitement parallèle d'informations en provenance du CPU et non plus d'afficher des images 3D performantes. En gros, ils réutilisent le même matériel que les shaders de rendu (vertex shader, geometry shader, pixel shader, etc…) mais utilisent la puissance disponible pour faire des calculs et du traitement de données.

Une carte graphique moderne est construite autour d'unités de calcul qui sont capables de traiter des vertices ou des pixels en parallèle pour générer au plus vite l'image demandée. Dans le cadre de DirectCompute, ces unités deviennent des threads capables d'exécuter du code utilisateur. Plus la carte est puissante et plus il y aura de threads disponibles : [Fig.1].

Une notion est toutefois importante à souligner lorsque l'on travaille avec DirectCompute : la gestion de la mémoire. En effet, comme nous pouvons le constater sur la figure 1, le GPU ne peut directement accéder à la mémoire centrale (ou alors en passant par l'AGP mais cela constitue une perte de performances trop importante). Il est donc nécessaire de garder à l'esprit que pour travailler, le GPU doit avoir ses données dans sa mémoire. Il faudra donc penser à faire une copie des données vers le GPU puis une recopie vers la RAM pour récupérer le résultat des traitements.

PORTABILITÉ ET SUPPORT MATÉRIEL

Pour faire fonctionner DirectCompute, il faut à minima disposer d'une carte graphique DirectX10. En effet, DirectX11 est capable de s'exécuter sur du matériel qui n'est pas directement compatible. Sur du matériel DirectX10, DirectCompute propose un sous-ensemble des fonctionnalités globales. Des limitations apparaissent, par exemple sur le nombre de threads mais l'ensemble reste largement utilisable.

Notre code d'initialisation va donc tenir compte de cette portabilité et va créer un device capable de fonctionner en 10 comme en 11 :

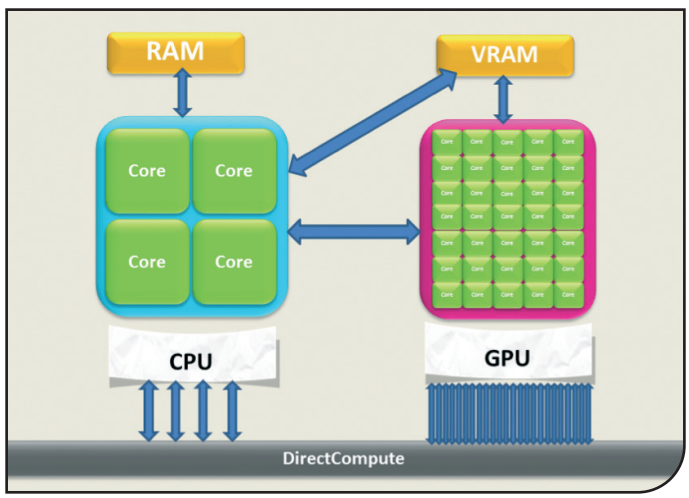


Figure 1. Fonctionnement interne de DirectCompute

```
        // Création de notre device (on accepte les cartes dx10 ou plus)
        FeatureLevel[] levels = {
              FeatureLevel.Level_11_0,
              FeatureLevel.Level_10_1,
              FeatureLevel.Level_10_0
            };
        // Creation du device
        device11 = new Device(DriverType.Hardware, DeviceCreation
Flags.None, levels);
        // Récupération du contexte immédiat
        immediateContext = device11.ImmediateContext;
        // Test d'éligibilité
        if (!device11.CheckFeatureSupport(Feature.ComputeShaders))
        {
          MessageBox.Show(@»Votre carte ne supporte pas Direct
Compute. Dommage :(«, @»Incompatibilité»);
          return;
```

CONCEPTION DU TRAITEMENT

Dans le cadre de notre exemple, nous allons écrire un shader qui va se charger de convertir une énorme chaîne de caractères en majuscules avec la contrainte supplémentaire de remplacer les espaces par des points. Ainsi, la chaîne « coucou le monde » va donc devenir « COUCOU.LE.MONDE ».

Nous réaliserons cette opération également avec le CPU pour obtenir une comparaison :

```
string data = File.ReadAllText(«data.txt»);

Stopwatch watch = new Stopwatch();
watch.Start();
byte[] result = new byte[data.Length];
for (int index = 0; index < data.Length; index++ )
{
  byte b = (byte)data[index];
  if (b == ' ')
    result[index] = (byte)'.';
  else if (b < 'a' || b > 'z')
    result[index] = b;
  else
    result[index] = (byte)(b - 32);
}

File.WriteAllText(«results.txt», Encoding.UTF8.GetString(result));
watch.Stop();
MessageBox.Show(watch.ElapsedMilliseconds.ToString());
```

CONCEPTION DU SHADER

Pour concevoir notre shader, nous allons nous limiter aux instructions permises par le modèle CS_4_0 (Compute Shader 4.0, la version 5.0 étant pour le matériel DirectX11).

Les Compute Shaders s'écrivent comme tous les shaders DirectX avec le langage HLSL (High Level Shader Language). DirectX11 a naturellement rajouté des nouvelles instructions au sein du langage pour apporter le support de DirectCompute.

Parmi ces ajouts nous pouvons notamment noter l'apparition de plusieurs types de ressources (autres que les textures) qui vont nous permettre d'accéder à nos données.

Dans le cadre de notre exemple, le type de ressources qui va nous intéresser est le **RWStructuredBuffer.** Cette ressource est un buffer en lecture/écriture (chose rarissime jusqu'alors) qui peut contenir des types de base ou des structures.

En ce qui concerne l'édition du shader en lui-même Visual Studio ne reconnaît pas nativement les fichiers au format .hlsl. Toutefois, un projet sur CodePlex permet de rajouter la coloration syntaxique sur ce type de fichiers : Il s'agit du projet NShader : http://nshader.codeplex.com/.

Pour la partie code qui sera exécutée par notre shader, il y a deux points à définir : la partie traitement et la partie déclaration des threads. En effet, notre shader doit définir combien de threads vont être utilisés. Ces threads peuvent être définis via un vecteur qui va permettre d'indexer (de numéroter) chaque thread lors de son exécution. Cette indexation servira de règle de calcul pour accéder aux ressources.

Par exemple, dans notre cas, chaque thread va prendre un caractère et va appliquer le traitement suivant : Si c'est une minuscule, je le transforme en majuscule et si c'est un espace je le transforme en point. Ainsi si notre chaîne contient 200 caractères, il suffit d'avoir 200 threads et le tour est joué.

Il existe toutefois des limites dans le nombre de threads total (768 en DirectX10 et 1024 en DirectX11). Dans notre cas nous allons faire des blocs de 256 threads et nous rappellerons plusieurs fois le shader pour couvrir toute la chaîne.

L'identification d'un thread est basée sur un vecteur car cela permet d'accéder à des ressources non linéaires comme des textures (donc avec un x,y) ou des volumes (donc avec un x,y et z). Notre shader va donc ressembler à ceci :

```
RWStructuredBuffer<int> Data : register( u0 )

//————————————————————————————
// Compute Shader
//————————————————————————————
[numthreads(256, 1, 1)]
void Process( uint3 Gid : SV_GroupID, uint3 GTid : SV_GroupThreadID )
{
 uint coord = GTid.x + 256 * Gid.x;

 uint data = Data[coord];
 uint result = 0;

 uint offset = 24;
 uint mask = 0xFF000000;
 for (int i = 0; i < 4; i++)
 {
   uint local = (data & mask) >> offset;

   // Si c'est un espace on met un .
   if (local == 32)
   {
     result += 46 << offset;
   }
   // Si ce n'est pas un caractere en minuscule on laisse tel quel.
   else if (local < 97 || local > 122)
   {
     result += local << offset;
   }
   // Sinon on transforme en majuscule
   else
     result += (local - 32) << offset;
   offset -= 8;
   mask = mask >> 8;
 }
 Data[coord] = result;

}
```

Nous voyons donc la déclaration du nombre de threads via le numthreads(256, 1,1) qui indique que l'on veut 256 threads en ligne.

Par la suite, la déclaration de notre traitement qui s'appelle Process et qui prend 2 paramètres : Gid et GTid qui sont respectivement affectés à des valeurs systèmes : SV_GroupID et SV_GroupThreadID (SV pour System Value). Le second paramètre va, pour chaque thread prendre la valeur d'index du thread dans la liste soit (0, 0, 0) jusqu'à (255, 0, 0) dans notre cas. Pour couvrir la taille globale de notre chaîne de caractères, il faudra appeler ce traitement plusieurs fois. Le premier paramètre (Gid) représentera l'index du groupe d'appels. Ainsi si l'on appelle 10 fois le traitement, Gid variera de (0, 0, 0) à (9, 0, 0). En effet, l'appel au traitement dans le code se fait via la méthode *Dispatch* à laquelle on indique un vecteur de récurrence, qui dans notre cas vaudra : (longueur de ma chaîne / 256, 1, 1). Ici encore nous sommes face à un vecteur pour indexer des données non linéaires.

```
// Lancement du calcul
immediateContext.Dispatch(Math.Max(1, dataLen / (256 * 4)), 1, 1);
```

On peut constater dans notre shader que l'accès aux données dans chaque thread se fait via le calcul suivant :

```
 uint coord = GTid.x + 256 * Gid.x;
```

On récupère l'index du thread (GTid.x avec y et z ne nous intéressant pas) auquel on ajoute l'index du groupe (Gid.x) multiplié par la largeur d'un groupe (256).

ANALYSE DU TRAITEMENT

La principale problématique dans notre code va venir du fait que les shaders ne savent pas manipuler les données de taille inférieure à un entier. Toutefois, ils savent faire des décalages de bits et cela va nous permettre de faire ce que l'on souhaite.
Lors du transfert des données depuis la mémoire centrale, nous récupérerons notre buffer Data rempli de uint issus de la conversion de la chaîne de caractères. Avec des décalages nous allons récupérer chaque byte et ainsi chaque thread traitera un groupe de 4 caractères. C'est aussi pour cela que le Dispatch prend en paramètre la valeur 256/4.

DÉFINITION DES BUFFERS DE TRAVAIL

En ce qui concerne le transit des informations, nous avons au final besoin de deux buffers : un pour envoyer les données vers le GPU et un pour les récupérer. Le second buffer est nécessaire car il faut spécifier un buffer lisible par le CPU (ce qui n'est pas le cas du premier car ce dernier sera stocké dans la VRAM) :

```
 // Buffers
 BufferDescription description = new BufferDescription();

 description.SizeInBytes = dataLen;
 description.BindFlags = BindFlags.UnorderedAccess | BindFlags.
ShaderResource;

 description.Usage = ResourceUsage.Default;
 description.OptionFlags = ResourceOptionFlags.StructuredBuffer;
 description.StructureByteStride = 4;

 elements = new Buffer(device11, description);

 // Vues pour écrire les données
 UnorderedAccessViewDescription uoViewDescription = new Unorder
edAccessViewDescription();

 uoViewDescription.Format = Format.Unknown;
 uoViewDescription.Dimension = UnorderedAccessViewDimension.Buffer;
uoViewDescription.ElementCount = dataLen / 4;
elementsView = new UnorderedAccessView(device11, elements, uoV
iewDescription);

 // Transfert des informations
 using (DataStream ds = new DataStream(datas, true, false))
 {
   DataBox box = new DataBox(0, 0, ds);
```

```
   immediateContext.UpdateSubresource(box, elements, 0);
 }
 // Buffer pour récupérer le résultat
 description = new BufferDescription();

 description.SizeInBytes = dataLen;
 description.Usage = ResourceUsage.Staging;
 description.CpuAccessFlags = CpuAccessFlags.Read;
 description.OptionFlags = ResourceOptionFlags.StructuredBuffer;
description.StructureByteStride = 4;

cpuBuffer = new Buffer(device11, description);
```

On notera ici que le buffer cpuBuffer porte le flag CpuAccessFlags.Read ce qui aura pour conséquence de mettre ce buffer dans la mémoire AGP.
Ainsi après notre dispatch, il nous suffit de venir copier le résultat dans cpuBuffer et d'aller lire les valeurs :

```
 // Récupération du résultat
 immediateContext.CopyResource(elements, cpuBuffer);
 DataBox box = immediateContext.MapSubresource(cpuBuffer, 0,
dataLen, MapMode.Read, MapFlags.None);
 byte[] results = new byte[dataLen];

 box.Data.Read(results, 0, dataLen);
 immediateContext.UnmapSubresource(cpuBuffer, 0);

 File.WriteAllBytes(filename, results);
```

CONCLUSION

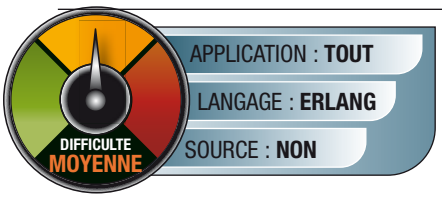
Hormis une mise en œuvre plus complexe, nous sommes face à des temps de l'ordre de la milliseconde pour DirectCompute contre des temps de l'ordre de la seconde pour le CPU.
On voit donc rapidement l'intérêt des Compute Shaders pour des gains forts en termes de temps d'exécution.
Toutefois, il est aussi important de noter que la mise en place est parfois difficile car il faut d'une part avoir un traitement parallèlisable (fortement) et d'autre part il faut que les contraintes (notamment en termes de taille et de temps de transfert des données) de DirectCompute, soient acceptables.

■ David Catuhe - *Bewise/Vertice*
blog : http://www.catuhe.com
site : http://www.bewise.fr I http://www.vertice.fr

Erlang, un langage à découvrir

1re partie

Erlang est un langage de programmation Open Source[1] fonctionnel, concurrent et distribué. Il est actuellement utilisé dans des logiciels tels que ejabberd (serveur Jabber/XMPP développé par ProcessOne)[2], Apache CouchDB (base de données orientée documents)[3], RabbitMQ (implémentation de AMQP récemment rachetée par VMWare)[4] ou Yaws (serveur HTTP)[5]. Parmi les acteurs utilisant Erlang, citons Facebook (système de chat), Amazon (simpleDB) ou GitHub (serveur egitd).



Erlang a été créé à l'origine par Ericson pour être utilisé dans les dispositifs réseaux. Il est par exemple utilisé pour faire fonctionner des équipements GPRS, accélérateurs SSL, répartiteurs de charge (certains boîtiers Althéon) ou switchs ATM, dont le fameux AXD301 d'Ericson, qui peut se targuer de fournir une disponibilité de 9 neuf (99.9999999%, soit 31 ms d'indisponibilité par an…) (6).

Erlang a depuis évolué vers un langage plus généraliste, mais nombre de ses spécificités résultent de son origine dans le monde des réseaux. En effet, les équipements réseaux doivent pouvoir supporter un grand nombre de connexions, ce qui implique de pouvoir gérer de nombreux threads. Erlang est effectivement *thread oriented* : les threads Erlang sont très légers et très simples à manipuler. Une demi-seconde suffit pour créer facilement 100 000 threads sur un PC de base, chaque thread occupant un minimum de mémoire. Ensuite, comme on l'a vu, le langage doit permettre de limiter au maximum les indisponibilités, même en cas d'erreurs dans le programme ou dans les données traitées. Erlang fournit pour cela un système d'arbre de supervision qui permet de relancer automatiquement une tâche qui se planterait, sans incidence sur le fonctionnement général de l'application. Il est également possible de mettre à jour le code de l'application à chaud, sans interruption de service.

Enfin, pour augmenter encore la disponibilité et la performance, les applications doivent pouvoir tourner sur plusieurs nœuds en parallèle. C'est pourquoi il est possible avec Erlang de manipuler les threads distants (c'est-à-dire tournant sur une autre machine physique) de la même manière que les threads locaux, de façon totalement transparente. Des mécanismes de réplication sont également fournis pour partager les données entre les nœuds. Il en résulte que les applications écrites en Erlang peuvent être clusterisées de façon pratiquement automatique.

D'un point de vue programmation, Erlang utilise le paradigme fonctionnel, ce qui implique une quasi-absence d'effets de bord (et donc de variables, d'affectations, de boucles, etc.). Dans le monde de la programmation parallèle, ne pas se soucier des effets de bord est un avantage énorme puisque l'on peut alors se passer des locks, mutex et autres mémoires partagées, qui sont habituellement un cauchemar pour les programmeurs utilisant des langages impératifs. Le code source d'un programme Erlang est compilé en bytecode et interprété par une machine virtuelle, nommée BEAM (pour *Bogdan/Björn's Erlang Abstract Machine*). Le compilateur Erlang et BEAM tournent sur la plupart des systèmes d'exploitation récents (Solaris, BSD, Linux, OSX, TRU64, IRIX, Windows NT/2000/2003/XP/Vista, VxWorks, iPhone, et probablement d'autres). Il est également possible de compiler certaines parties de code Erlang en code natif en utilisant un outil nommé HIPE (High Performance Erlang).

Erlang est distribué avec un ensemble de bibliothèques de développement appelé OTP pour Open Telecom Platform. OTP fournit la plupart des fonctions de base permettant de manipuler des structures de données de plus ou moins haut niveau, les protocoles réseaux courants, mais dispose également d'une base de données répartie (mnesia), d'un serveur HTTP, d'un compilateur ASN.1 et de bien d'autres choses encore. La plupart des applications utilisant Erlang utilisent en fait Erlang/OTP.

INSTALLATION

La plupart des distributions Linux disposent d'un paquet Erlang, qui reste la méthode la plus simple pour installer l'environnement de développement complet. Il est également possible de le compiler depuis les sources disponibles sur le site officiel(7), à l'aide des commandes suivantes :

```
./configure --enable-smp-support –enable-kernel-poll
make
sudo make install
```

Un installeur binaire pour Windows est également disponible sur ce même site. Pour vérifier qu' Erlang est correctement installé, lancer le shell Erlang à l'aide de la commande erl, ou sélectionner Démarrer -> Erlang OTP -> Erlang sous Windows :

```
% erl
Erlang R13B01 (erts-5.7.2) [source] [64-bit] [smp:2:2] [rq:2]
[async-threads:0] [kernel-poll:false]

Eshell V5.7.2  (abort with ^G)
1>
```

(1) *Erlang est publié sous licence EPL (Erlang Public licence) qui est basée sur la Mozilla Public License (MPL).*
(2) *http://www.process-one.net/en/ejabberd/*
(3) *http://couchdb.apache.org/*
(4) *http://www.rabbitmq.com/*
(5) *http://yaws.hyber.org/*
(6) *http://www.pragmaticprogrammer.com/articles/erlang.html*
(7) *http://erlang.org/download.html*

La première ligne permet de vérifier, entre autres, la version de Erlang/OTP utilisée, la version de l'environnement run-time (ERTS, Erlang Run-Time System), le nombre de bits utilisés en fonction de l'OS et le nombre de CPU détectés.

PREMIERS PAS DANS LE SHELL

Une fois le shell lancé, il est possible de l'utiliser comme une simple calculatrice :

```
1> 1+1.
2
2> 2-2.
0
3> 3*3.
9
4> 4/4.
1.00000
```

La première remarque à faire ici est qu'une « phrase » Erlang se termine toujours par un point.
Il est également possible d'appeler les fonctions des bibliothèques fournies par OTP. La syntaxe est module:fonction(paramètres) :

```
5> calendar:is_leap_year(2000).
true
6> calendar:is_leap_year(2002).
false
7> calendar:last_day_of_the_month(2004,2).
29
```

Les parenthèses sont obligatoires même si la fonction ne demande pas d'arguments :

```
8> calendar:local_time().
{{2007,2,7},{17,28,37}}
```

LES TYPES DE DONNÉES

Entiers et réels

```
1> 0.
0
2> -1.
-1
3> 12345678901234567890123456789012345678901234567890*12345678901234567890.
152415787532388367503429357750342935775019051998750190521 00
```

Erlang n'a pas de limite prédéfinie pour la taille des valeurs, la limite pratique étant la mémoire disponible. Une notation alternative existe pour les entiers, en utilisant le code ASCII des caractères :

```
1> $a.
97
2> $A.
65
```

Il s'agit uniquement de sucre syntaxique, les résultats sont équivalents avec la notation classique des entiers.
Pour les réels, il est possible d'utiliser la notation classique :

```
1> 3.14116.
3.14116
2> -12.3456.
-12.3456
3> 4/2.
2.00000
4> 12345678901234567890.12345678912345678912345678123456789.
21345678902345678.
1.52416e+35
```

ou la notation exponentielle :

```
1> 1234.56.
1234.56
2> 1.23456E3.
1234.56
3> 1.23456e3.
1234.56
```

Atomes

Les atomes sont des constantes représentées sous la forme d'une suite de caractères. Ils commencent par une lettre minuscule, suivie uniquement par des lettres, chiffres, et le caractère souligné OU sont entourés de simples quotes.

```
1> true.
true
2> atome1.
atome1
3> atome_2.
atome_2
4> atOme3.
atOme3
5> 'Atome4'.
'Atome4'
6> 'atome 5 !'.
'atome 5 !'
```

Listes

Les listes peuvent contenir tous types de données hétérogènes, y compris d'autres listes. Erlang utilise des listes chaînées comme structure de donnée interne.

```
1> [1, 2, 3].
[1,2,3]
2> [ok, error, true, false].
[ok,error,true,false]
```

Listes imbriquées :

```
3> [[1, 2], [3, 4, 5]].
[[1,2],[3,4,5]]
```

Les éléments d'une liste peuvent être de types différents (ici atomes et entier) :

```
4> [un, 2, trois].
[un,2,trois]
```

Liste vide :

```
5> [].
[]
```

Il n'existe pas de type spécifique pour les chaînes de caractères. Pour les représenter, on utilise donc des listes d'entiers représentant les codes ASCII des caractères de la chaîne. Pour simplifier l'écriture, une notation alternative des listes est utilisée pour représenter les chaînes de caractères, en utilisant des doubles quotes :

```
1> "Hello World !".
"Hello World !"
2> "abc".
"abc"
3> [$a, $b, $c].
"abc"
4> [97, 98, 99].
"abc"
```

Les trois derniers exemples ci-dessus sont trois notations différentes de la même valeur (une chaîne de trois entiers) et sont donc rigoureusement identiques.

Tuples

Les *tuples* (appelé *record* ou *struct* dans d'autres langages) permettent de regrouper des données hétérogènes en un seul terme.
Un ensemble composé du nom et prénom, de l'âge et du sexe peut être regroupé dans un tuple pour représenter une personne :

```
1> {nom, prenom, age, sexe}.
{nom,prenom,age,sexe}
2> {"Dupond", "Jean", 42, homme}.
{"Dupond","Jean",42,homme}
3> {"Martin", "Marie-Charlotte", 69, femme}.
{"Martin","Marie-Charlotte",69,femme}
```

Les tuples sont également utilisés pour typer des valeurs, comme un code de retour, par exemple :

```
4> {ok, 123}.
{ok,123}
5> {error, "Not Found"}.
{error,"Not Found"}
```

LES VARIABLES

Nous avons vu dans la présentation du langage qu'il y avait peu d'effets de bord en Erlang et notamment pas de variables. En fait elles existent tout de même, mais ne peuvent être « assignées » qu'une fois. Lorsque l'on assigne une valeur à une variable, elle est liée (bind) à cette valeur. Une variable liée le reste tout au long sa vie, le lien ne peut pas être modifié.
Le nom d'une variable doit commencer par une lettre majuscule (pour la différencier d'un atome).
Par défaut, une variable n'est pas liée, on ne peut pas l'utiliser :

```
1> A.
** 1: variable 'A' is unbound **
```

Le signe égal permet de lier une variable :

```
2> A=3.
3
3> A.
3
```

Si on essaye de lier la variable avec une autre valeur, on obtient une erreur :

```
4> A=4.

=ERROR REPORT==== 7-Feb-2007::18:06:09 ===
Error in process <0.30.0> with exit value: {{badmatch,4},[{erl
_eval,expr,3}]}

** exited: {{badmatch,4},[{erl_eval,expr,3}]} **
```

LA CORRESPONDANCE DE MOTIF

L'erreur que l'on obtient ci-dessus est *badmatch*. L'opérateur égal est en fait un opérateur de correspondance de motif (match) et non pas d'égalité. Lorsqu'il est utilisé, Erlang va essayer de faire correspondre le type et la valeur à droite de l'opérateur avec l'expression de gauche. Par exemple :

```
5> A=3.
3
```

Ici, l'entier 3 matche avec la valeur liée précédemment à A, donc le match est réussi, et la valeur retournée est celle de l'expression de droite. Lorsque l'on matche une valeur avec une variable non liée, alors la valeur est liée avec la variable.
Il est possible de matcher des termes complexes :

```
1> {A1, A2} = {123, 456}.
{123,456}
2> A1.
123
3> A2.
456
4> {A2, A3} = {456, A1}.
{456,123}
5> A3.
123
6> {ok, B} = {ok, [1, 2, 3]}.
{ok,[1,2,3]}
7> B.
[1,2,3]
```

Après avoir effectué ces opérations, quel serait d'après vous le résultat du match suivant ?

```
8> {A3, A2} = {A1, 456.0}.
```

ACCUMULATION EN TÊTE DE LISTE

L'opérateur pipe (|) dans une liste permet de séparer le premier élément d'une liste (head) du reste de la liste (tail).
Il peut être utilisé pour construire des listes par accumulation :

```
1> L = [2, 3, 4].
[2,3,4]
2> L2 = [1 | L].
[1,2,3,4]
```

Il est également très utile pour matcher :

```
3> [Head | Tail] = L2.
[1,2,3,4]
4> Head.
1
5> Tail.
[2,3,4]
```

MODULES

Jusqu'ici, nous n'avons utilisé que le shell Erlang. Essayons maintenant d'écrire un programme, à savoir le classique « Hello, Word ». Dans un fichier nommé hello.erl, écrivons les quatre lignes suivantes :

```
-module(hello).
-export([world/0]).

world() ->
 io:format("Hello, World !~n", []).
```

Un fichier .erl est un fichier source Erlang qui correspond à un module. La valeur de la directive -module doit correspondre au nom du fichier (sans l'extension), -export permet de déclarer les fonctions accessibles depuis l'extérieur du module (son API, donc). Les deux dernières lignes définissent une fonction world/0 (d'arité 0). La fonction io:format/2 permet d'afficher du texte formaté, comme le printf du C. Pour compiler ce programme dans un shell, il faut utiliser la fonction c (le fichier .erl doit être dans le répertoire courant :

```
1> c(hello).
{ok,hello}
```

Il est également possible de compiler en ligne de commande à l'aide de erlc :

```
erlc hello.erl
```

Le résultat est un fichier hello.beam, qui contient le bytecode Erlang. Il est maintenant possible d'utiliser notre module depuis le shell, sous réserve que le fichier .beam soit dans le répertoire courant :

```
1> hello:world().
Hello, World !
ok
```

LES FONCTIONS

Dans notre exemple précédent, nous avons défini une fonction. Un nom de fonction doit respecter les mêmes règles que pour un atome. Le corps d'une fonction peut comporter plusieurs expressions, séparées par des virgules, et se termine par un point. La valeur de retour de la fonction est la valeur de la dernière expression.

Lors de l'appel d'une fonction, Erlang va effectuer un match entre la valeur des paramètres d'appel et l'expression de la définition de la fonction. Ainsi, pour écrire une fonction qui retourne le premier élément d'une liste, on peut écrire :

```
premier([Head | Tail]) ->
   Head.
```

Si on essaye d'appeler

```
premier([1, 2, 3])
```

une opération de match similaire à

```
[Head | Tail] = [1, 2, 3]
```

est effectuée lors du passage des paramètres.

Il est possible de définir plusieurs clauses pour la même fonction. La première clause qui permet de réussir un match est utilisée. Les clauses sont séparées par des point-virgules :

```
factoriel(0) -> 1;
factoriel(N) -> N * factoriel(N-1).
```

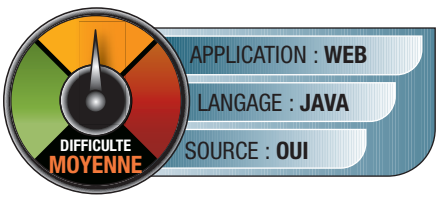
CONCLUSION

Nous n'avons vu dans ce premier article que les éléments de base qui permettent d'installer, de tester, de compiler et de faire tourner un programme Erlang basique. Nous avons également vu les principaux types de base du langage, mais nous n'avons pas encore abordé l'une de ses spécificités les plus intéressantes, à savoir la manipulation des threads. Nous y consacrerons donc un second article, qui tentera de montrer la puissance d'Erlang dans ce domaine, en termes de performances et de facilité d'utilisation.

■ Jérôme **Sautret**
ProcessOne

Apache Pivot, l'autre RIA

Apache Pivot est un framework de développement d'applications Internet riches (RIA) qui a pris de plus en plus de place depuis sa promotion au rang de « Top Level Project », au sein de la fondation Apache.



Dans cet article, nous allons essayer de développer une simple application client/serveur de gestion de contacts.

PARTIE CLIENTE

L'écran principal se composera d'une liste de contacts et d'un bouton pour ajouter un contact à la liste. Un clic sur une entrée de la liste permettra à l'utilisateur de modifier les détails du contact [Fig.1 et 2].
Les deux écrans sont conçus déclarativement dans les deux fichiers table.xml et addContact.xml.
Extrait (correspondant au tableau de contacts) de table.xml

```
<TableView wtkx:id="tableView"
    styles="{includeTrailingVerticalGridLine:true}">
    <columns>
        <TableView.Column name="firstname" width="100"
            headerData="Prenom" />
        <TableView.Column name="lastname" width="100"
            headerData="Nom" />
        <TableView.Column name="email" width="60"
            headerData="E-Mail" />
        <TableView.Column name="phone" width="60"
            headerData="Téléphone" />

    </columns>
</TableView>
```

Extrait (correspondant au formulaire) de addContact.xml

```
<Form styles="{rightAlignLabels:true}" wtkx:id="submitform">
    <sections>
        <Form.Section>
            <Label Form.label="ID" textKey="id" />
            <BoxPane Form.label="Nom">
                <TextInput textKey="lastname" />
            </BoxPane>
            <BoxPane Form.label="Prénom">
                <TextInput textKey="firstname" />
            </BoxPane>
            <BoxPane Form.label="Téléphone">
                <TextInput textKey="phone" />
            </BoxPane>
            <BoxPane Form.label="Adresse E-mail">
                <TextInput textKey="email" />
            </BoxPane>
        </Form.Section>
    </sections>
</Form>
```

Notre classe Contact a les propriétés ID (id), Prénom (firstName), Nom (lastName), E-mail (email) et téléphone (phone). Noter que dans le composant TableView de table.xml, la propriété 'name' de chaque colonne correspond à la propriété de la classe Contact correspondante, et que dans chaque TextInput du formulaire d'ajout/modification de addContact.xml, la propriété textKey correspond aussi à la propriété de la classe Contact correspondante.
À part la classe Contact (modèle de données), la partie cliente se compose de 4 classes :

- Une classe Contacts qui joue le rôle du contrôleur (implémentant trois interfaces d'écoute d'événements)
- Une classe TableContactsWindow (vue) qui représente la fenêtre principale contenant la liste des contacts
- Une classe EditContactWindow (vue) qui représente la fenêtre d'ajout/modification d'un contact
- Une classe RemoteContactModel (modèle) qui sera responsable de la communication avec le serveur

Nos trois interfaces d'écoute d'événements (implémentées par le contrôleur) sont :

- ContactUpdateListener pour répondre à deux événements : demande (par l'utilisateur) d'ajout d'un contact et demande (par l'utilisateur) d'édition d'un contact.
- ContactSubmitRequestListener pour répondre à l'événement de demande (par l'utilisateur) de sauvegarde d'un contact.
- ContactAccessListener pour répondre à deux événements.

LES VUES

La classe TableContactsWindow garde une référence sur un écouteur d'événements ContactUpdateListener (c'est la table de contacts qui déclenche les événements associés) et a deux propriétés : le tableau de contacts et la fenêtre contenant le tableau. Le constructeur prend en paramètre l'écouteur et demande à Pivot de parser table.xml pour construire la fenêtre et la table :

```
WTKXSerializer wtkxSerializer = new WTKXSerializer();
try {
  window = (Window)wtkxSerializer.readObject(this, «table.xml»);
} catch (IOException e) {
  e.printStackTrace();
} catch (SerializationException e) {
  e.printStackTrace();
}
table = (TableView)wtkxSerializer.get(«tableView»);
```

Ensuite, on ajoute un gestionnaire pour l'événement de clic sur une ligne de la table (modification d'un contact) et pour l'événement de clic sur le bouton 'Ajouter' :

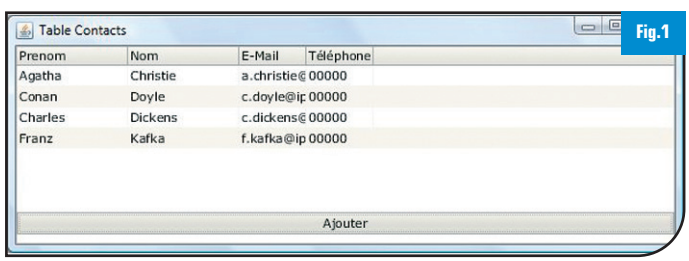


Fig.1

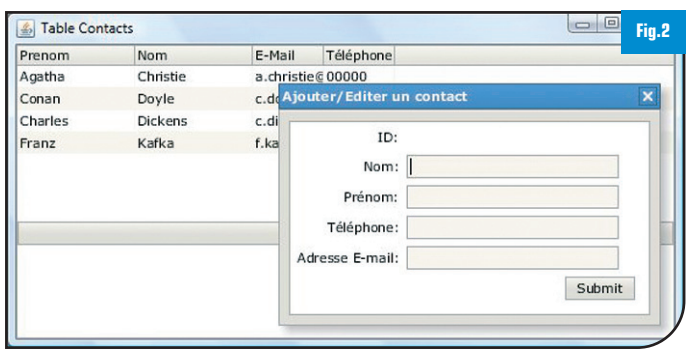


Fig.2

```
//Ajoute un gestionnaire d'événement pour le clic sur une ligne du tableau
table.getComponentMouseButtonListeners().add(new ComponentMouseButtonListener.Adapter() {
 @SuppressWarnings(«unchecked»)
 @Override
 public boolean mouseClick(Component component,
   org.apache.pivot.wtk.Mouse.Button button, int x, int y, int count) {
   List<Contact> contacts = (List<Contact>)table.getTableData();
   //Récupère l'indice de la ligne cliqué
   int index = table.getRowAt(y);
   //Notifie le contrôleur de la demande d'édition d'un contact
   controller.editContactRequest(contacts.get(index), TableContactsWindow.this);
       return false;
     }
   });

   PushButton addButton  = (PushButton)wtkxSerializer.get(«addButton»);

   //Ajoute un gestionnaire d'événement pour le clic sur le bouton 'Ajouter'
   addButton.getButtonPressListeners().add(new ButtonPressListener() {
     @Override
     public void buttonPressed(Button button) {
       //Notifie le contrôleur du clic sur le bouton 'Ajouter'
       controller.addContactRequest(TableContactsWindow.this);
     }
   });
```

Elle admet aussi une méthode setContacts pour remplir la table avec une liste de contacts :

```
 public void setContacts(List<Contact> result) {
   //Met à jour la table des contacts
   table.setTableData(result);
 }
```

Notons ici, qu'en appelant les méthodes setTableData et getTableData, Apache Pivot se charge du mapping entre les propriétés de la classe Contact et les colonnes du tableau. La classe EditContactWindow garde une référence sur un écouteur d'événements ContactSubmitRequestListener (c'est la fenêtre d'édition qui déclenche l'événement de demande de soumission) et a les propriétés suivantes :

- la fenêtre de tableau de contacts associée (à laquelle ajouter ou depuis laquelle modifier le contact)
- la fenêtre d'édition
- le formulaire d'édition
- l'objet Contact en cours d'ajout/d'édition

Le constructeur prend en paramètre le contrôleur et la fenêtre de tableau de contacts associée, demande à Pivot de parser addContact.xml pour construire la fenêtre d'édition et le formulaire :

```
   WTKXSerializer wtkxSerializer = new WTKXSerializer();
   try {
     editWindow = (Window)wtkxSerializer.readObject(this,
     «addContact.xml»);
   } catch (IOException e) {
     e.printStackTrace();
   } catch (SerializationException e) {
     e.printStackTrace();
   }

   form = (Form)wtkxSerializer.get(«submitform»);
```

Ensuite, on ajoute un gestionnaire pour l'événement de clic sur le bouton de soumission du formulaire :

```
 PushButton submit = (PushButton)wtkxSerializer.get(«submitButton»);
 submit.setAction(new Action() {
   @Override
   public void perform() {
   //Met à jour l'objet contact avec les valeurs des champs du formulaire
   //Apache Pivot se charge de faire l'association
   //entre les propriétés de la classe Contact et les champs du formulaire
   form.store(contact);
   //Notifie le contrôleur de la soumission du formulaire
   listener.contactSubmit(contact,EditContactWindow.this);
   //Ferme la fenêtre d'edition
   editWindow.close();
   }
 });
```

La classe EditContactWindow admet aussi une méthode setContact pour mettre à jour l'objet Contact en cours d'édition :

```
public void setContact(Contact contact) {
 this.contact = contact;
 //Les champs du formulaire se remplissent avec les infos du contact
 //Apache Pivot se charge de faire l'association
 //entre les propriétés de la classe Contact et les champs du formulaire
 form.load(contact);
}
```

En appelant les méthodes *load* et *store* de l'objet *Form*, Pivot se charge de mapper les propriétés de la classe Contact avec les champs du formulaire.

LE MODÈLE

À ce niveau de l'article, nous n'allons pas détailler la classe RemoteContactModel, qui incarne le modèle dans notre application. Sachons seulement que cette classe implémente l'interface ContactModel, et est chargée de deux opérations :

- Récupérer la liste des contacts (opération à la fin de laquelle elle déclenche l'événement contactsRetrieved)
- Poster un objet Contact pour ajout ou pour mise à jour (opération à la fin de laquelle elle déclenche l'événement contactSaved)

LE CONTRÔLEUR

La classe Contacts joue le rôle de contrôleur en implémentant les 3 interfaces d'écoute et en gardant une référence vers la vue TableContactsWindow et le modèle ContactModel. Pour être notre point d'entrée de l'application, elle implémente l'interface org.apache.pivot.wtk.Application. Pivot appellera alors la méthode startup :

```
//Crée la fenêtre de table de contacts
   tcw = new TableContactsWindow(this);
```

```
    //Instancie le model
    model = new RemoteContactModel(this);
    //Demande au modèle de récupérer la liste des contacts
    model.retrieveAllContacts();
    //Affiche la fenêtre sur l'écran principal
    tcw.getWindow().open(display);
```

Dans cette méthode, on instancie une fenêtre de tableau de contacts, on la remplit avec la méthode updateTable (qui ne fait que demander au modèle de récupérer la liste), et enfin on l'affiche sur l'écran principal (l'objet display qu'on reçoit en paramètre). La classe Contacts, jouant le rôle de contrôleur, est responsable de gérer les événements :

1. Demande d'édition d'un contact : instanciation d'une fenêtre d'édition et lui passer le contact à éditer

```
  public void editContactRequest(Contact contact, TableContacts
Window source) {
    //Crée une fenêtre d'édition de contact
    EditContactWindow ecw = new EditContactWindow(this,source);
    //Met l'objet Contact cliqué pour édition
    ecw.setContact(contact);
    //Affiche la fenêtre
    ecw.show();
  }
```

2. Demande d'ajout d'un contact : instanciation d'une fenêtre d'édition et lui passer un nouvel objet contact « vide »

```
  public void addContactRequest(TableContactsWindow source) {
    //Crée un nouvel objet Contact «vide»
    Contact newContact = new Contact(null, «», «», «», «»);
    //Crée une fenêtre d'édition
    EditContactWindow ecw = new EditContactWindow(this,source);
    //Met le nouveau contact pour édition
    ecw.setContact(newContact);
    //Affiche la fenêtre d'édition
    ecw.show();
  }
```

3. Demande de soumission d'un contact : demande au modèle de sauvegarder (poster) l'objet

```
public void contactSubmit(Contact contact, EditContactWindow source) {
    model.saveContact(contact);
  }
```

4. Chargement de la liste des contacts : met à jour la table des contacts avec la nouvelle liste.

```
public void contactsRetrieved(List<Contact> contacts) {
    tcw.setContacts(contacts);
  }
```

5. Sauvegarde d'un contact : demande au modèle un rechargement de la liste des contacts

```
public void contactSaved() {
    model.retrieveAllContacts();
  }
```

LA PARTIE SERVEUR

Du côté serveur, on a deux composants essentiels :
- Une Servlet qui gère les requêtes de la partie cliente
- Un DAO pour récupérer et sauvegarder les contacts

La servlet communique avec le DAO pour sauvegarder un objet posté par le client, ou pour récupérer une liste de contacts à renvoyer au client. Pour simplifier les choses, nous avons opté pour une implémentation en mémoire du DAO, en tant que map dont les clés sont les ID et les valeurs sont les contacts associés.

```
public class MemoryContactDAO implements ContactDAO {

  private static Map<Long, Contact> data = new HashMap<Long, Contact>();
  static Long lastId;

  public List<Contact> getAll() {
    List<Contact> contacts = new ArrayList<Contact>();
    for(Contact c:data.values()){
      contacts.add(c);
    }
    return contacts;
  }

  @Override
  public Contact getById(Long id) {
    return data.get(id);
  }

  @Override
  public void save(Contact contact) {
    if(contact.getId() != null){
      data.put(contact.getId(), contact);
    }else{
      lastId ++;
      contact.setId(lastId);
      data.put(lastId, contact);
    }

  }
```

L'INTERACTION CLIENT/SERVEUR

Apache Pivot dispose d'une API pour lancer des requêtes HTTP depuis la partie cliente. Deux classes de cet API sont utilisées dans notre classe modèle de la partie cliente.
Pour récupérer la liste des contacts, on lance une requête HTTP de type GET, et une fois exécutée avec succès, on notifie le contrôleur de la réponse reçue.

```
public void retrieveAllContacts() {
    //Crée une requête GET
    GetQuery getQuery = new GetQuery(HOST,PORT, «/»+APP_NAME+»
/Contact»,false);
    //Affecte le sérialiseur de contacts
    getQuery.setSerializer(SerializerFactory.getContactSerializer());
    //Lance la requête en affectant un écouteur pour gérer le retour
 de la réponse
    getQuery.execute(new TaskListener<Object>() {
      @Override
      public void taskExecuted(Task<Object> task) {
        //task.getResult() = Réponse Objet désérialisé assumé
 en tant que liste de contacts
        //On notifie le controller de la liste récupérée
```

```
        controller.contactsRetrieved(((List<Contact>)task.get
Result()));
      }
      @Override
      public void executeFailed(Task<Object> task) {
        //Ne fait rien du tout si erreur
      }

    });
  }
```

Pour sauvegarder (poster) un objet contact, on lance une requête HTTP de type POST, et une fois exécutée avec succès, on notifie le contrôleur.

```
public void saveContact(Contact contact) {
    //Crée une requête POST
    PostQuery post = new PostQuery(HOST,PORT, «/»+APP_NAME+»
/Contact»,false);
    //Affecte le sérialiseur de contacts
    post.setSerializer(SerializerFactory.getContactSerializer());
    //Affecte l'objet Contact soumis (qui sera sérialisé)
    post.setValue(contact);
    //Lance la requête
    post.execute(new TaskListener<URL>() {

      @Override
      public void executeFailed(Task<URL> arg0) {
        //Ne fait rien du tout si erreur
      }

      @Override
      public void taskExecuted(Task<URL> arg0) {
        //On notifie le controller que la requête s'est bien déroulée
        controller.contactSaved();
      }
    });


  }
```

Vous avez peut-être remarqué dans le code qu'on affecte à notre objet requête un sérialiseur. Les sérialiseurs d'Apache Pivot peuvent faire l'objet d'un article qui leur est dédié. Sachez qu'Apache Pivot ne fait pas de différence entre sérialiseurs et désérialiseurs, et que dans ce contexte (requête GET), il s'agit plutôt d'un désérialiseur pour permettre à Pivot de « comprendre » la réponse renvoyée par le serveur (task.getResult()). Dans cette application, on a utilisé le BinarySerializer, qui ne fait qu'une sérialisation (pour les objets postés) et une désérialisation (pour les objets reçus) en binaire. Évidemment, ce type de sérialiseur est rarement pratique, nous l'avons choisi juste par simplicité et parce que c'est nous qui développons la partie cliente ainsi que la partie serveur. Par contre, si nous voulons utiliser notre partie serveur pour exposer des services consommés par d'autres types de clients (Web, Application .NET…), le BinarySerializer est un choix à proscrire. Ce que Pivot apporte dans cette partie, c'est la sérialisation et la désérialisation transparente pour le développeur qui ne manipule que des objets. Remarquez que task.getResult() nous a retourné une liste de contacts, et que pour poster un objet contact, on n'a eu à faire que post.setValue(contact). Apache Pivot dispose aussi d'une API pour traiter les requêtes HTTP du côté serveur. En effet, Pivot nous offre la classe QueryServlet. Dans notre application, la servlet de la partie serveur hérite de QueryServlet. Lorsqu'on hérite de QueryServlet, on doit implémenter (redéfinir) :

- La méthode newSerializer qui retourne le sérialiseur/désérialiseur que Pivot utilise pour sérialiser les objets retournés au client, et désérialiser les objets reçus du client
- Une méthode par type de requête HTTP à gérer

Dans notre cas, nous utiliserons le BinarySerializer comme retour de la méthode newSerializer : c'est ainsi que le client a sérialisé ses données, c'est ainsi donc que le serveur devrait les désérialiser. De plus, nous devrons redéfinir la méthode doGet et doPost pour gérer les deux types de requêtes que le client envoie.

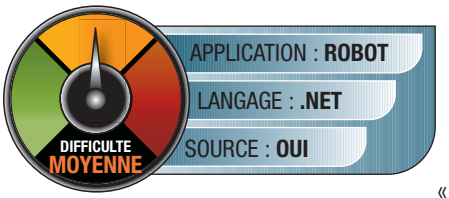
```
protected Object doGet() throws ServletException, ClientException {

  //Récupère le DAO
  ContactDAO dao = DAOFactory.getContactDao();
  //Cas où la requête porte sur tous les contacts
  List<Contact> all = dao.getAll();
  org.apache.pivot.collections.List<Contact> result = new org.
apache.pivot.collections.ArrayList<Contact>();
  //Transforme une java.util.List en l'implémentation List de Pivot
  for(Contact c:all){
    result.add(c);
  }
  //Retourne la liste
  //Apache Pivot se chargera par la suite de sérialiser la liste
  //et de l'inclure dans une réponse HTTP
  return result;
}

protected URL doPost(Object value) throws ServletException,
ClientException {
  //Apache Pivot s'est déjà chargé de désérialiser l'objet envoyé
  //dans la requête POST et nous l'a passé en paramètre
  Contact contact = (Contact)value;
  //Récupère le DAO
  ContactDAO dao = DAOFactory.getContactDao();
  //Enregistre(ajoute ou met à jour) le contact envoyé
  dao.save(contact);
  try {
    String app = this.getContextPath();
    return new URL(«http://localhost:8080»+app+»/Contact»);
  } catch (MalformedURLException e) {
    e.printStackTrace();
  }
  return null;
}
```

Remarquez dans les deux méthodes, qu'on ne manipule pas les objets HTTPServletRequest, ni HTTPServletResponse. Les signatures des méthodes doGet et doPost sont différentes de celles des servlets « normales » (javax.servlet.http.HttpServlet).

■ Hamdi **Douss**
Expert Java/JEE chez IP-Tech, SSII tunisienne partenaire offshore des éditeurs de logiciels français (www.iptech-offshore.com).

Microsoft Robotics Developer Studio : L'intégration des différents services 3e partie

Après les deux précédents numéros (128 et 131), nous poursuivons notre exploration et programmation de Robotics Developer Studio. Dans cette ultime partie nous allons aborder les services. Bon robot !



Comment intégrer ces nouveaux services dans une infrastructure plus large ? La réponse n'est pas simple, car elle dépend du « méta-service » que l'on veut proposer : parfois on veut juste que tous les services démarrent en même temps, mais on les interroge séparément. Dans d'autres cas, on préfère créer une interface unique qui permet de monitorer les différents services en même temps (sur un PC ou sur un mobile par exemple). Dans le premier cas, il suffit de créer un nouveau manifest avec le DSS Manifest Editor, dans lequel on crée les partenariats vers tous les services que l'on veut démarrer.

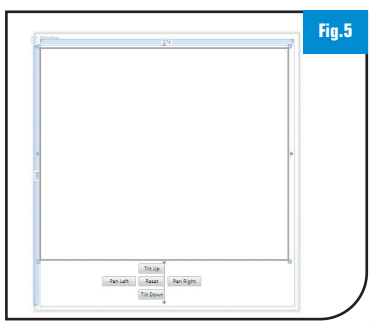
C'est un peu basique, nous voulons pousser plus loin : le deuxième cas permet d'implémenter une logique de contrôle spéciale (déclenchement d'une alarme et allumage des lumières si mouvement sur la caméra par exemple) ou une interface unifiée (pour voir à la fois l'état de la caméra, des lumières et du chauffage). C'est ce cas que nous allons traiter. Nous allons donc créer à l'aide du wizard un nouveau service, déclarer les partenariats vers les services Webcam et PanTiltCamControl et ensuite créer une interface spéciale qui nous permettra à la fois de voir l'image de la caméra, et d'en contrôler le mouvement. Le wizard passé, voici le code qui est généré automatiquement et qui définit les partenariats de notre « méta-service »

```
/// <summary>
/// PanTiltCamControlService partner
/// </summary>
[Partner("PanTiltCamControlService", Contract = ptcc.Contract.
Identifier, CreationPolicy = PartnerCreationPolicy.UseExisting
OrCreate)]
ptcc.PanTiltCamControlOperations _panTiltCamControlServicePort = new
 ptcc.PanTiltCamControlOperations();
ptcc.PanTiltCamControlOperations  _panTiltCamControlService
Notify = new ptcc.PanTiltCamControlOperations();

/// <summary>
/// WebCamService partner
/// </summary>
[Partner("/WebcamService", Contract = webcam.Contract.Identifier,
CreationPolicy = PartnerCreationPolicy.UseExistingOrCreate)]
webcam.WebCamOperations _webcamServicePort = new Microsoft.
Robotics.Services.WebCam.Proxy.WebCamOperations();
webcam.WebCamOperations _webcamServiceNotify = new Microsoft.
Robotics.Services.WebCam.Proxy.WebCamOperations();
```

On voit que pour chaque partenariat ont été créés deux ports de communication : un port principal par lequel on enverra des commandes, et un port de notification par lequel on recevra les mises à jour d'état de ces services. Ajoutons maintenant une fenêtre à notre projet. Pour cela, un simple clic droit sur le projet dans la vue solution nous ouvrira le menu contextuel permettant l'ajout d'un nouvel élément : ensuite il convient de choisir WPF User Control (mais vous pouvez choisir les WinForms si vous le souhaitez ;))

L'interface, simpliste pour les besoins de l'exercice, sera une simple fenêtre en WPF constituée d'une image (reçue depuis la caméra) et de 5 boutons (Haut, Bas, Gauche, Droite, Reset) correspondant aux 5 messages qu'on peut envoyer. Attention, à ce stade, une petite feinte est requise : il faut tout de suite modifier dans le XAML et dans le code-behind l'héritage de notre interface, qui ne doit pas être un UserControl mais une fenêtre (Window).

Voici l'interface, qui est extrêmement simple : quelques glisser/déposer des contrôles sur le designer de form, et le tour est joué : [Fig.5].



Une fois l'interface prête, il faut prévoir la façon de communiquer entre le service et l'interface graphique. Cette communication sera bidirectionnelle : il faut pouvoir envoyer des données du service vers la fenêtre (comme par exemple l'image de la caméra) mais également pouvoir transformer les événements des boutons de l'interface en commandes pour notre Service PanTiltCamControl. Dans le sens interface graphique -> Service, la méthode recommandée est de passer l'instance du service en argument dans le constructeur du formulaire, ce qui permettra à ce dernier d'appeler, à l'appui d'un bouton, une méthode interne du service, cette méthode transformant le nom du bouton en commande à envoyer sur le port du service PanTiltCamControl. Voici le code des constructeurs :

```
PanTiltCamGUIService _service;

public PanTiltCamGUI()
{
    InitializeComponent();
}

internal PanTiltCamGUI(PanTiltCamGUIService service) : this()
{
    _service = service;
}
```

Et voici le code des handlers de clics sur les boutons :

```
private void btnUp_Click(object sender, RoutedEventArgs e)
```

```
{
    _service.Button_Click("up");
}

private void btnDown_Click(object sender, RoutedEventArgs e)
{
    _service.Button_Click("down");
}

private void btnLeft_Click(object sender, RoutedEventArgs e)
{
    _service.Button_Click("left");
}

private void btnRight_Click(object sender, RoutedEventArgs e)
{
    _service.Button_Click("right");
}

private void btnReset_Click(object sender, RoutedEventArgs e)
{
    _service.Button_Click("reset");
}
```

Dans le service, la méthode interne qu'on appelle, Button_Click, est la suivante :

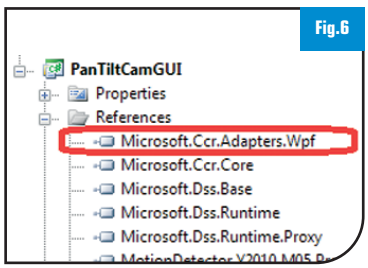
```
internal void Button_Click(string cmd)
{
    ptcc.PanTiltMovement RequestedMove;
    switch (cmd)
    {
        case "up":
            RequestedMove = ptcc.PanTiltMovement.TiltUp;
            break;
        case "down":
            RequestedMove = ptcc.PanTiltMovement.TiltDown;
            break;
        case "left":
            RequestedMove = ptcc.PanTiltMovement.PanLeft;
            break;
        case "right":
            RequestedMove = ptcc.PanTiltMovement.PanRight;
            break;
        case "reset":
            RequestedMove = ptcc.PanTiltMovement.ResetPOV;
            break;
        default:
            RequestedMove = ptcc.PanTiltMovement.ResetPOV;
            break;
    }

    Activate(
        Arbiter.Choice(
          _panTiltCamControlServicePort.PanTilt(new ptcc.Pan
TiltRequest() { Move = RequestedMove } ),
            success => { /* Do nothing */ },
            fault =>
            {
                _wpfServicePort.Invoke(() => _userInterface.
ShowFault(fault));
            }
        )
    );
}
```

On utilise un « Choice » pour pouvoir récupérer et traiter les messages d'erreur et les afficher sur l'interface graphique. La méthode ShowFault s'exécute dans le contexte WPF (car on utilise Invoke), en voici le contenu :

```
/// <summary>
/// Displays a fault message
/// </summary>
/// <param name="fault">fault</param>
internal void ShowFault(W3C.Soap.Fault fault)
{
    var error = "Erreur!";

   if (fault.Reason != null && fault.Reason.Length > 0 && !string.
IsNullOrEmpty(fault.Reason[0].Value))
    {
        error = fault.Reason[0].Value;
    }

    MessageBox.Show(
        this,
        error,
        Title,
        MessageBoxButton.OK,
        MessageBoxImage.Error);
}
```

Nous avons donc maintenant une communication entre le service et l'interface graphique, qui sont contenus dans le même projet. Reste à démarrer cette interface graphique au même moment que le service : cela passe par référencer le bon « Adapter », en l'occurrence ici le CCR WPF Adapter, à rajouter à la fois en référence et dans les using du projet. [Fig.6]

Fig.6



Ensuite, on déclare dans le corps du service un nouveau port de service WPF à travers lequel nous communiquerons avec la fenêtre, ainsi qu'une instance de la fenêtre en elle-même :

```
ccrwpf.WpfServicePort _wpfServicePort;
PanTiltCamGUI _userInterface;
```

Enfin, à partir de la méthode Start(), on démarre l'interface graphique :

```
/// <summary>
/// Service start
```

```
/// </summary>
protected override void Start()
{

    //
    // Add service specific initialization here
    //
    SpawnIterator(Initialize);
}

IEnumerator<ITask> Initialize()
{

    #region Create UI
    _wpfServicePort = ccrwpf.WpfAdapter.Create(TaskQueue);
    var runWindow = _wpfServicePort.RunWindow(() => new Pan
TiltCamGUI(this));
    yield return (Choice)runWindow;

    var exception = (Exception)runWindow;
    if (exception != null)
    {
        LogError(exception);
        StartFailed();
        yield break;
    }

    _userInterface = (Window)runWindow as PanTiltCamGUI;

    #endregion

    #region Subscribe to webcam notifications
    var subscribe = _webcamServicePort.Subscribe(_webcamService
Notify, typeof(webcam.UpdateFrame));
    yield return (Choice)subscribe;

    var fault = (Fault)subscribe;
    if (fault != null)
    {
        LogError(fault);
        StartFailed();
        yield break;
    }
    #endregion

    StartComplete();
}

void StartComplete()
{
    base.Start();

    MainPortInterleave.CombineWith(
        Arbiter.Interleave(
            new TeardownReceiverGroup(),
            new ExclusiveReceiverGroup(
              Arbiter.Receive<webcam.UpdateFrame>(true, _web
camServiceNotify, UpdateFrameHandler)
```

```
                ),
            new ConcurrentReceiverGroup()
        )
    );
}
```

Ce code nous montre une finesse classique de Robotics Developer Studio qui consiste, en utilisant un IEnumerator et l'instruction yield, à écrire du code de façon séquentielle mais non bloquante : songez que chaque fois qu'on appelle yield, on redonne en fait la main à CCR pour exécuter ses tâches et on revient automatiquement dans le code juste après le yield quand cette tâche est finie. Une fois habitué à cette notation, la lisibilité est grandement améliorée !
On voit dans cette initialisation la souscription aux notifications en provenance de la caméra, et l'association de ces notifications à un handler : en fait la caméra nous préviendra à chaque fois qu'une nouvelle frame est prête : après, à nous, depuis le handler de cette notification, de la télécharger et éventuellement de la traiter : en l'occurrence il faut mettre à jour la fenêtre WPF !

```
public void UpdateFrameHandler(webcam.UpdateFrame update)
{
    SpawnIterator(update, UpdateFrameHandlerIterator);
}


public IEnumerator<ITask> UpdateFrameHandlerIterator(webcam.Up
dateFrame update)
{

    Fault fault = null;
    webcam.QueryFrameRequest request = new webcam.QueryFrame
Request();
    webcam.QueryFrameResponse response = null;

    request.Format = ImageFormat.Jpeg.Guid;
    request.Size = new Microsoft.Robotics.PhysicalModel.Proxy
.Vector2(640, 480);

    yield return Arbiter.Choice(
        _webcamServicePort.QueryFrame(request),
        delegate(webcam.QueryFrameResponse success)
        {
            response = success;
        },
        delegate(Fault f)
        {
            fault = f;
        }
    );

    if (fault != null)
    {
        yield break;
    }


   _state.LastFrameStream = new MemoryStream(response.Frame, false);
    _wpfServicePort.Invoke(() =>
    {
        try
        {
            _state.LastFrameStream.Seek(0, SeekOrigin.Begin);
            _userInterface.image1.Source = BitmapFrame.Create
(_state.LastFrameStream, BitmapCreateOptions.None, BitmapCache
Option.OnLoad);

        }
        catch (Exception ex)
        {
            LogError(ex.Message);
        }
        });
}
```

Ce dernier bout de code nous montre comment interroger le service webcam pour récupérer une image, toujours en utilisant yield dans un IEnumerator afin de ne pas bloquer CCR dans un thread en attendant une réponse, et également comment mettre simplement à jour l'image dans la fenêtre WPF à partir du stream reçu.

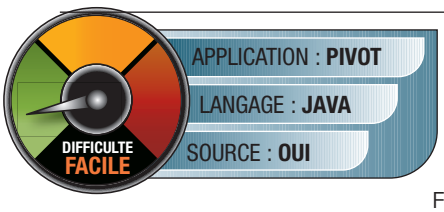
CONCLUSION

Nous avons vu à travers un exemple simple de « robotisation » d'une caméra toutes les notions permettant de construire une architecture sous forme de services avec Microsoft Robotics Developer Studio : réutilisation d'un service existant, développement d'un nouveau service, et intégration de ces services dans une interface commune. Le code est bien entendu disponible, avec un petit cadeau pour ceux d'entre vous qui souhaiteraient aller plus loin : j'ai implémenté en plus de ce qui est décrit dans cet article un service de détection de mouvement simpliste, avec lequel nous créerons un partenariat qui permettra à la fois d'afficher ce mouvement dans l'interface graphique (il pourrait par exemple lever une alarme) mais également d'orienter la caméra vers le mouvement en question, par exemple pour suivre une personne passant dans le champ de la caméra. Cadeau Bonux !

■ Pierre **Cauchois**

Créer des applications RIA en Java avec Pivot

Les partisans de Java apprécieront le framework Pivot, qui leur permettra de développer des applications avec leur langage favori. Découverte.



Les Applications Internet Riches sont à la mode et les frameworks pour en développer se multiplient. Nous avons AIR du côté de Flash et d'Adobe, Silverlight du côté de Microsoft. Quid du côté de Java ? il existe au moins deux frameworks. D'abord Jspresso (www.jspresso.org, qui est une pile Jav/Flex/AJAX et nécessite Tomcat comme serveur d'applications. De l'avis de votre serviteur, JSpresso est très élaboré et complet, mais assez lourd à manier. Nous avons aussi (parmi d'autres) Pivot avec lequel nous nous familiariserons aujourd'hui. Pivot est un projet de la fondation Apache. Il est relativement léger, très facile à apprendre et agréable à manipuler. Les applications sont faciles à déployer. Une application RIA Pivot est une applet dans le navigateur ou une application Java classique sur votre bureau. Nul besoin de Tomcat. N'importe quel serveur Web fera l'affaire. En ce qui concerne les prérequis, il est nécessaire de connaître Java, bien entendu, ainsi que Javascript et d'avoir quelques notions de XML.

1 LES OUTILS

Nous allons donc travailler avec Pivot (pivot.apache.org) dans sa version 1.4 au moment de la rédaction de cet article. Peu d'outils sont nécessaires. Il est toutefois important de noter que Java 1.6 update 10 est la version de Java minimum pour travailler avec Pivot. Pour les essais des applets nous avons utilisé le serveur Apache d'un EasyPhp sous Windows, et nous avons écrit le code des exemples avec Ganymède (Eclipse 3.4.2). Les sources des exemples sont disponibles sur notre site.

2 VUE D'ENSEMBLE DE LA PLATE-FORME

Les applications Pivot sont basées sur l'architecture modèle-vue-contrôleur (MVC). Du côté du modèle nous trouvons une librairie centrale (Core) qui contient des collections génériques. Côté contrôleur, nous avons toute la panoplie de composants possibles et imaginables et Côté vue, nous avons les Skin qui définissent le rendu des composants. En ce qui concerne le langage, on travaille avec Java qui est à la base de la plate-forme, même si au final, ce n'est pas lui qui sera le plus présent dans une application Pivot, et avec un langage de script compatible avec la JVM. Ici Rhino, le Javascript supporté par Java 1.6. Si, selon les préférences du développeur, Java peut être très peu présent dans le code d'une application Pivot, est à la base du runtime, et c'est lui et ses capacités d'introspection qui rendent les applications Pivot si faciles à écrire.

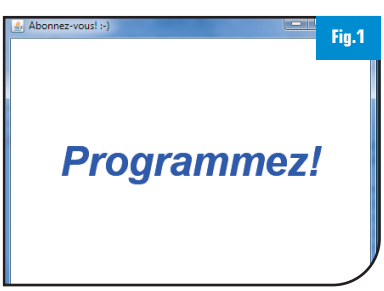


Fig.1

Notre première application Pivot, d'une originalité folle.

3 HELLO WORLD RICHE

Une application Pivot est à la fois une application classique et une applet. Pour cette raison, quand une applet Java étend la classe Applet (ou JApplet) et redéfinit, selon les besoins, les méthodes init, start, stop et destroy, une application Pivot implémente l'interface org.apache.pivot.wtk.Application et définit des méthodes semblables à celle de Applet, comme ceci :

Application Pivot	Applet java	Description
startup	Init	Invoquée au démarrage de l'application.
suspend	Stop	Invoquée quand l'utilisateur quitte la page de l'applet.
resume	Start	Invoquée quand l'utilisateur revient à la page de l'applet. Contrairement à start qui est invoquée automatiquement après init, resume n'est pas invoquée automatiquement après startup
shutdown	Destroy	Invoquée à la fermeture de l'application

C'est du moins la théorie. La pratique nous réserve des surprises. Nous le verrons en essayant notre première application dont voici le code : [Fig.1].

```
// HelloWorld.java
package com.programmez.fred.demospivot;

import java.awt.Color;
import java.awt.Font;

import org.apache.pivot.collections.Map;
import org.apache.pivot.wtk.Application;
import org.apache.pivot.wtk.DesktopApplicationContext;
import org.apache.pivot.wtk.Display;
import org.apache.pivot.wtk.HorizontalAlignment;
import org.apache.pivot.wtk.Label;
import org.apache.pivot.wtk.VerticalAlignment;
import org.apache.pivot.wtk.Window;

public class HelloWorld implements Application {
  Window window;
```

```
@Override
public void startup(Display display,
    Map<String, String> properties) {
  System.out.println(«Coup d'envoi»);
  Label label = new Label();
  label.setText(«Programmez!»);
  label.getStyles().put(
    «font», new Font(«Times», Font.BOLD|Font.ITALIC, 48));7
  label.getStyles().put(«color», Color.BLUE);
  label.getStyles().put(«horizontalAlignment»,
    HorizontalAlignment.CENTER);
  label.getStyles().put(«verticalAlignment»,
    VerticalAlignment.CENTER);

  window = new Window();
  window.setContent(label);
  window.setTitle(«Abonnez-vous! :-)»);
  window.setMaximized(true);
  window.open(display);
}

@Override
public boolean shutdown(boolean optional) {
  System.out.println(«Fin de partie»);
  if (window != null) {
    window.close();
  }
  return false;
}

@Override
public void suspend() {
  System.out.println(«Temps mort»);
}

@Override
public void resume() {
  System.out.println(«Reprise»);
}

public static void main(String[] args) {
  DesktopApplicationContext.main(HelloWorld.class, args);
}
}
```

Ce code est sans difficulté. Nous y voyons que l'emploi de composants ne dépaysera pas le programmeur familiarisé avec Swing ou AWT. Il y a toutefois quelques remarques à faire. L'initialisation se fait dans la méthode startup comme dit plus haut. Le runtime passe un objet Display en argument à startup. Cet objet encapsule le conteneur racine de notre application. Dans le corps de startup, à la dernière ligne, nous attachons un objet Windows qui est le containeur de plus haut niveau pour l'utilisateur. Autrement dit, Windows est le premier objet visible à l'utilisateur. Tant que l'objet Windows n'est pas attaché à l'objet Display, il n'est pas pleinement opérationnel, ou réalisé, comme l'on dit parfois avec ce genre de librairie. Ainsi s'il est possible d'ajouter conteneurs et composants à volonté à Windows, demander une opération d'affichage échouerait en provoquant une exception. Nous remarquons encore, dans le corps de shutdown qui est la méthode invoquée lorsque l'application se termine, qu'il incombe au programmeur de détruire l'objet Windows. Les exemples sur le site de Pivot sont écrits ainsi. Toutefois, la raison à cela est obscure et le bien fondé de la destruction manuelle de l'objet Window n'est pas du tout évident. Enfin, les essais montrent que si l'on demande à chaque méthode, *startup, suspend, resume,* et *shutdown,* d'écrire un message sur la console, seules *startup* et *shutdown* le font lorsque l'application est lancée en mode autonome, hors navigateur. Dans le navigateur, ainsi que nous le verrons plus loin, les choses sont encore différentes, et dans tous les cas non conformes à ce qu'indique la documentation. Tel est du moins le comportement observé par votre serviteur. Si vous avez constaté un comportement différent, nous vous invitons à nous le signaler sur notre forum, dans la discussion consacrée à cet article.

4 DÉPLOIEMENT

En utilisation autonome, nous devons lancer notre application en lançant le runtime et en passant la classe principale de notre application à celui-ci. Bien évidemment, nous devons veiller à placer les fichiers Jar de Pivot dans le CLASSPATH. Par exemple (sur une seule ligne de commande) :

```
java -cp tous-les-jar-requis.jar
org.apache.pivot.wtk.DesktopApplicationContext
com.programmez.fred.demospivot.HelloWorld
```

Dans le navigateur, notre application est avant tout une applet ordinaire que le navigateur lancera s'il trouve une balise applet dans la page HTML, par exemple comme ceci :

```
<applet
  code=»org.apache.pivot.wtk.BrowserApplicationContext$HostApplet»
  archive=»lib/pivot-core-1.4.jar,lib/pivot-wtk-1.4.jar,lib/
pivot-wtk-terra-1.4.jar»
  width=»400» height=»280»>
  <param name=»application_class_name»
    value=»com.programmez.fred.demospivot.HelloWorld»>
</applet>
```

Bien remarquer que la classe utilisée pour lancer le runtime est différente de celle utilisée en mode autonome (attribut code). Là encore, les jars de Pivot doivent être accessibles au navigateur (attribut archive). Enfin dans l'élément param, c'est bien *application_class_name* qui doit être la valeur de l'attribut name, et non *applicationClassname,* comme indiqué par erreur dans la documentation au moment de la rédaction de cet article. En alternative à la balise applet, il est sans doute pertinent de passer par l'utilitaire de déploiement de Sun. Le lecteur trouvera des exemples dans le code source des pages HTML des démos de Pivot. Résumons-nous. Nous avons une applet déployée, ce qui pour l'instant ne casse pas trois pattes à un canard, et présente un inconvénient de taille : la taille des jars de Pivot, qui sont assez longs à télécharger, et donc retardent le démarrage de l'applet d'autant. Si l'on considère qu'en bidouillant un peu il n'est pas difficile d'écrire du code Java pour avoir une application qui soit aussi une applet, on se demande si l'on est

gagnant à travailler avec Pivot, surtout qu'une applet un tant soit peu complexe demandera probablement à être signée. La réponse sera clairement oui, dès que l'on aura fait connaissance avec son atout maître : WTKX

5 WTKX

WTKX est un langage à la XML que l'on utilisera, de préférence à Java, pour construire les interfaces des applications. Ce travail est facile, et cela offre la possibilité de modifier une interface sans devoir recompiler du code. Le côté Java et le côté XML peuvent communiquer ensemble, et côté XML, Javascript fait des merveilles. Voyons ceci avec quelques exemples, et commençons par réécrire notre Hello World. Côté Java d'abord (code partiel)

```
import org.apache.pivot.wtkx.WTKXSerializer;
// import etc...

public class HelloWorldWTKX implements Application {
 private Window window;

 @Override
 public void startup(Display display, Map<String, String> properties)
   throws Exception {
   WTKXSerializer wtkxSerializer = new WTKXSerializer();
   window = (Window)wtkxSerializer.readObject(
     this, «helloworld.xml»);
   window.open(display);
 }
// etc.
}
```

Dans ce code Java allégé, nous ne faisons que charger un fichier à la XML avec une classe magique WTKXSerialiser. Voici son contenu :

```
<Window title=»Hello WTKX!» maximized=»true»
 xmlns:wtkx=http://pivot.apache.org/wtkx
 xmlns=»org.apache.pivot.wtk»>

 <content>
   <Label text=»Programmez!»
     styles=»{font:'Arial bold italic 48', color:'#0000ff',
     horizontalAlignment:'center',
     verticalAlignment:'center'}»/>
 </content>
</Window>
```

Nous voyons une balise Label. Le runtime en fera un composant Label. Cette balise a des attributs. D'où sortent-ils ? Tout simplement Pivot utilise les capacités d'introspection de Java sur les composants qui sont autant de Beans. Un composant a-t-il des accesseurs tels que *getText* et *setText* ? Alors *text* est un attribut valide. Si seul l'accesseur *getText* était présent alors *text* serait en lecture seule.

C'est tout simple. Regardons maintenant attentivement l'attribut *styles*. Cet attribut existe car Label possède un accesseur *getStyles* (hérité de Component). Cet accesseur, nous dit la Javadoc de Pivot, est un dictionnaire. Nous comprenons donc pourquoi, dans le code ci-dessus l'attribut *styles* se voit affecté une structure au format JSON qui n'est rien d'autre qu'un dictionnaire. Bien. Mais comment connaître quelles sont les clés de ce dictionnaire ? Nous avons parlé plus haut de Skin. Ceux-ci sont définis par des classes Java qui sont parallèles aux classes des composants. Ainsi si Label est une classe de composant, on trouvera à coup sûr dans la javadoc de Pivot une classe *LabelSkin*. Cette classe comme toutes les classes de Skin est un Bean.

Et, à nouveau, en regardant la présence des accesseurs dans la javadoc, on en déduit les clés de dictionnaires que nous cherchions.

Un dernier mot à propos de cet exemple: il faut avoir à l'esprit que notre fichier XML, dans le cas d'une utilisation en applet, n'est pas chargé par le navigateur, mais par la JVM qui doit être en mesure de le trouver. Déposer le fichier XML à côté du fichier .class de l'application est une réponse simple à cette question.

6 UN HELLO WORLD MIS EN FORME

Enrichissons un peu notre Hello World [Fig.2]. Notre application présente maintenant deux Label. Pivot offre des classes de mises en forme (layout) pour le positionnement relatif des composants. Nous n'entrons pas dans le détail de ceux-ci. Nous renvoyons le lecteur à la documentation. La méthode startup de notre exemple HelloWorldLayout.java utilise une BoxPane comme gestionnaire de mise en forme. Un code finalement assez verbeux.

Fig.2

Pivot offre des gestionnaires de mise en forme pour le positionnement relatif des composants.

Faisons un aparté: pour l'occasion, le lecteur, s'il essaie ce code complet dans son navigateur, verra que les méthodes *suspend, resume,* et même cette fois *shutdown* ne sont pas invoquées. Revenons à WTKX et laissons Java trop verbeux. Si nous reprenons HelloWorldWTKX, il suffit de lui faire charger un autre fichier XML pour obtenir le même résultat. Voici le fichier :

```
<Window title=»Hello WTKX!» maximized=»true»
 xmlns:wtkx=http://pivot.apache.org/wtkx
 xmlns=»org.apache.pivot.wtk»>

 <content>
   <BoxPane orientation=»vertical»
   styles=»{horizontalAlignment:'center',
        verticalAlignment:'center'}»>
     <Label text=»Programmez!»
       styles=»{font:'Arial bold 48',
          color:'#0000ff'}» />
     <Label text=»Le magazine du développement!»
       styles=»{font:{bold:true, italic:true,
         size:24, name:'Arial'}, color:'#ff0000'}» />
   </BoxPane>
 </content>
</Window>
```

Ce qui est finalement à la fois plus court et plus clair.

7 RÉAGIR AUX ACTIONS DE L'UTILISATEUR

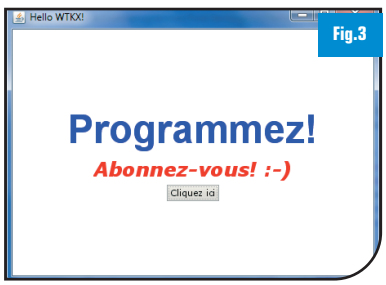


Fig.3

Un bouton fonctionnel, avec un peu de XML et un peu de Javascript

Là encore, si cela peut se faire côté Java, nous préférons le faire côté XML. Toujours avec HelloWorld WTKX, nous chargeons un nouveau fichier XML. Notre Hello World est cette fois enrichi d'un bouton pleinement fonctionnel [Fig.3].

```
<Window title=»Hello WTKX!» maximized=»true»
  xmlns:wtkx=http://pivot.apache.org/wtkx
  xmlns=»org.apache.pivot.wtk»>
  <content>
  <BoxPane orientation=»vertical»
  styles=»{horizontalAlignment:'center',
      verticalAlignment:'center'}»>
    <Label text=»Programmez!»
      styles=»{font:'Times bold 48', color:'#0000ff'}» />
    <Label wtkx:id=»monlabel»
      text=»Le magazine du développement!»
      styles=»{font:{bold:true, italic:true,
        size:24, name:'Times'}, color:'#ff0000'}» />
    <PushButton buttonData=»Cliquez ici»>
      <buttonPressListeners>
        <wtkx:script>
          function buttonPressed(button) {
            monlabel.setText(«Abonnez-vous! :-)»);
          }
        </wtkx:script>
      </buttonPressListeners>
    </PushButton>
  </BoxPane>
    </content>
</Window>
```

Ce code illustre comment écrire, en Javascript, le gestionnaire d'événement du bouton. Bien remarquer l'attribut wtkx:id du Label dont le contenu sera modifié lors du clic sur le bouton. Il est possible de faire encore plus court en plaçant le code Javascript comme valeur d'un attribut. Voici un extrait de helloworld-button-listener-attribute, disponible au complet sur notre site

```
<Label wtkx:id=»monlabel» text=»Le magazine du développement!»
  styles=»{font:{bold:true, italic:true, size:24, name:'Times'},
  color:'#ff0000'}» />
<PushButton
  buttonData=»Cliquez ici»
  ButtonPressListener.buttonPressed=»monlabel.setText('Abonnez-
vous! :-)')» />
```

8 COLLABORATION JAVA ET JAVASCRIPT

Nous en arrivons au meilleur de Pivot : la classe magique WTKXSerialiser permet à Java et Javascript d'échanger des objets entre eux et dans les deux sens (code complet dans HelloworldScripting.java sur notre site) :

```
private Window window;
private String prog = «Progammez!»;
private String mag = «mag: Chaine non initialisee»;
private String texte = «texte: Chaine non initialisee»;

@Override
public void startup(Display display, Map<String, String> properties)
  throws Exception {
  WTKXSerializer wtkxSerializer = new WTKXSerializer();
  System.out.println(mag);
  System.out.println(texte);
  wtkxSerializer.put(«mag», mag);
  wtkxSerializer.put(«prog», prog);
  window = (Window)wtkxSerializer.readObject(this,
    «helloworld-scripting.xml»);
  mag = (String)wtkxSerializer.get(«mag»);
  texte = (String)wtkxSerializer.get(«texte»);
  System.out.println(«mag: « + mag);
  System.out.println(«texte: « + texte);
  window.open(display);
}
```

Ce code passe trois chaînes, *prog*, *mag*, et *texte* au côté XML/Javascript, et récupère *mag* et *texte* après exécution. Le code XML (fichier complet helloworld-scripting.xml sur notre site) contient peu de nouvelles choses.
Pour l'essentiel, nous y voyons comment définir un secteur de code Javascript et aussi comment incorporer le code d'un fichier .js :

```
<?language javascript?>
<wtkx:script>
  importClass(java.lang.System);

  function buttonClicked() {
    System.out.println(«» + texte);
    monlabel.setText(texte);
  }
</wtkx:script>
<wtkx:script src=»helloworld-scripting.js»/>
```

Voici pour terminer le code du fichier .js :

```
importClass(java.lang.System);
System.out.println(«[Javascript]: « + mag);
mag = «Le magazine du développement»;
texte = «Abonnez-vous! :-)»;
```

Pour conclure, nous dirons qu'écrire des applications RIA avec Pivot est facile et même agréable.
Sorti depuis peu de l'incubateur Apache, Pivot mérite de figurer en bonne place entre AIR et Silverlight.

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

DÉVELOPPEZ VOTRE SAVOIR-FAIRE

Economisez 16,45 €

Abonnement 1 an au magazine

11 numéros par an
49 €
au lieu de 65,45 € tarif au numéro
Tarif France métropolitaine

Programmez ! *est le magazine du développement*

Langage et code, développement web, carrières et métier : Programmez !, c'est votre outil de veille technologique.

Pour votre développement personnel et professionnel, abonnez-vous à Programmez !

www.programmez.com

- **Abonnement Intégral : 1 an au magazine + archives sur Internet et PDF : 59 €**
  Tarif France métropolitaine
- **Abonnement PDF / 1 an : 30 €** - *Tarif unique*
  Inscription et paiement exclusivement en ligne
  www.programmez.com
- **Abonnement Etudiant : 1 an au magazine : 39 €**
  (au lieu de 65,45 € tarif au numéro)
  Offre France métropolitaine

+ Abonnement INTÉGRAL

ACCÈS ILLIMITÉ aux **ARCHIVES du MAGAZINE** pour **0,84€ par mois !**

Cette option est réservée aux abonnés pour 1 an au magazine, quel que soit le type d'abonnement (Standard, Numérique, Etudiant). Le prix de leur abonnement normal est majoré de 10 € (prix identique pour toutes zones géographiques). Pendant la durée de leur abonnement, ils ont ainsi accès, en supplément, à tous les anciens numéros et articles/ dossiers parus.

OUI, *je m'abonne*

Vous pouvez vous abonner en ligne et trouver tous les tarifs www.programmez.com

PROGRAMMEZ

- ☐ **Abonnement 1 an au magazine : 49 €** (au lieu de 65,45 € tarif au numéro) *Tarif France métropolitaine*
- ☐ **Abonnement Intégral : 1 an au magazine + archives sur Internet et PDF : 59 €** *Tarif France métropolitaine*
- ☐ **Abonnement Etudiant : 1 an au magazine : 39 €** (joindre copie carte étudiant) *Offre France métropolitaine*

☐ M. ☐ Mme ☐ Mlle Entreprise : ..... Fonction : .....

Nom : ..... Prénom : .....

Adresse : .....

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

Tél : ..... E-mail : .....

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

A remplir et retourner sous enveloppe affranchie à :
Programmez ! - Service Abonnements - 22 rue René Boulanger - 75472 Paris Cedex 10.
abonnements.programmez@groupe-gli.com

Offre limitée, valable jusqu'au 31 août 2010

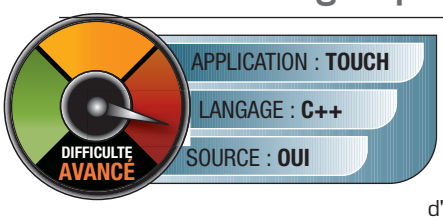
Le renvoi du présent bulletin implique pour le souscripteur l'acceptation pleine et entière de toutes les conditions de vente de cette offre.
Conformément à la loi Informatique et Libertés du 05/01/78, vous disposez d'un droit d'accès et de rectification aux données vous concernant.
Par notre intermédiaire, vous pouvez être amené à recevoir des propositions d'autres sociétés ou associations.
Si vous ne le souhaitez pas, il vous suffit de nous écrire en nous précisant toutes vos coordonnées.

PROG 132

La programmation multitouch sous Windows 7

1re partie

Une des nouveautés majeures de Windows 7 est le support des interfaces tactiles. Nous découvrons comment mettre en œuvre les API afin de construire des applications qui obéissent au doigt et pas encore à l'œil.



1 LE PRINCIPE

Un écran tactile est un périphérique qui combine les fonctionnalités d'affichage d'un écran classique aux fonctionnalités d'un périphérique de pointage comme une souris. Sous Windows, les impulsions électriques émises par une souris sont transformées par le système en messages postés à l'intention de l'application concernée. Lorsque l'application reçoit ces messages, un mécanisme fait que pour chaque message, une fonction de rappel dite procédure de fenêtre est invoquée de manière asynchrone. C'est dans cette procédure de fenêtre que l'application réagira à ces messages. Puisqu'un écran tactile est lui aussi un dispositif de pointage, le principe reste avec lui identique, même si l'on se doute que la teneur des messages et leurs traitements sont un peu plus complexes. L'API proposée par Windows 7 pour travailler avec tout cela est constituée de deux parties. Une partie "Gesture" (ou geste, ndlr) qui ne saura gérer les touchers que d'un ou deux doigts, mais saura déduire quels gestes ont été effectués par l'utilisateur, et une partie "Touch", qui supporte autant de points de contact que le périphérique peut en fournir, mais demandera plus de travail au programmeur qui devra déduire quels ont été les mouvements effectués. Une application travaille avec l'une ou l'autre de ces parties, jamais avec les deux simultanément.

2 LES OUTILS

Vous devez posséder un Windows 7, et télécharger le SDK pour Windows 7 sur le site. Dans ce SDK vous trouverez des en-têtes et des librairies pour développer des applications en C/C++. N'importe quel compilateur fera l'affaire. Le tout est de le faire pointer sur les répertoires contenant les en-têtes et les librairies.

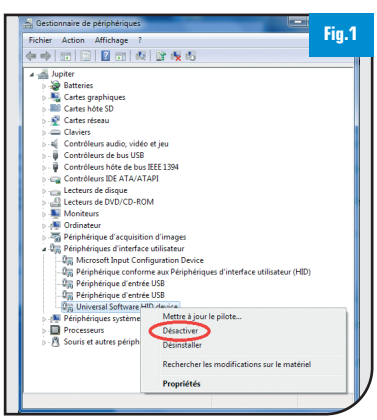


Fig.1

Le driver de l'émulateur Multi-Touch Vista doit être activé manuellement.

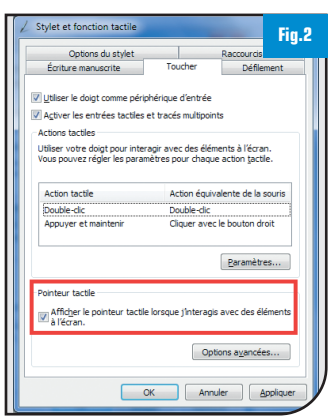


Fig.2

Pour pouvoir travailler avec l'émulateur, l'affichage des pointeurs doit être activé.

Tous les exemples accompagnant cet article, et disponibles sur notre site, ont été écrits avec Visual Studio 2008. Le SDK Windows contient aussi des exemples, et bien sûr une documentation avec quelques petites erreurs. Au départ l'API MultiTouch est conçue pour être programmée en C/C++. Le framework .Net 4, en bêta au moment de la rédaction de cet article, contiendra des classes pour travailler avec le multitouch. En attendant il existe une très bonne librairie, pour le framework 3.5, à cette adresse http://code.msdn.microsoft.com/ WindowsTouch. Nous verrons comment l'utiliser dans la suite de cet article. Enfin, il faut un périphérique tactile, ce dont tout le monde ne dispose pas… Heureusement, il existe un émulateur. Pour expérimenter avec celui-ci, vous devez connecter à votre machine autant de souris que vous souhaitez simuler de doigts. Et vous sentir une âme de jongleur ;-)

3 INSTALLER L'ÉMULATEUR MULTI-TOUCH VISTA

Si on ne dispose pas de périphérique tactile, on téléchargera cet émulateur à http://multitouchvista.codeplex.com/. La première chose à faire est d'installer le driver en lançant, depuis une console avec les droits de l'administrateur, la commande :

```
"Install driver.cmd"
```

Cette commande se trouve dans le sous-répertoire Driver de l'archive de l'émulateur. Le driver n'étant pas signé, vous recevrez un avertissement. Confirmez l'installation malgré tout. Ceci fait, vous devez vous livrer à une petite manipulation dans le gestionnaire de périphériques, afin d'activer ce driver. Dans les périphériques d'interface utilisateur, cliquez avec le bouton droit de la souris sur Universal Software HID device. Sélectionnez alors 'Désactiver' dans le menu contextuel, comme illustré [Fig.1]. Répétez immédiatement l'opération en sélectionnant cette fois 'Activer'. Cette opération effectuée, l'applet 'Stylet et Fonctions Tactile' apparaît dans le panneau de configuration. Cliquez dessus, et sous l'onglet 'Toucher', cochez la case 'Afficher le pointeur tactile lorsque j'interagis avec des éléments à

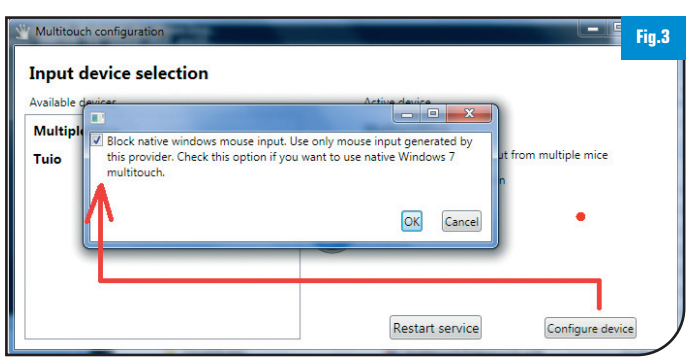


Fig.3

Faites disparaître le pointeur natif et vous voilà dans l'univers du tactile.

l'écran' comme illustré [Fig.2]. Puis sous l'onglet défilement, cochez toutes les cases. Il est maintenant temps de lancer l'émulateur. Les souris (qui sont autant de doigts) avec lesquelles vous souhaitez travailler doivent être dès à présent connectées à la machine. À la racine de l'arborescence de l'émulateur, commencez par lancer l'application Multitouch.Service.Console.exe en cliquant dessus. Puis lancez MultitouchDriverConsole.exe. A ce stade vous devrez sans doute utiliser les touches de votre clavier pour y parvenir. Enfin lancez Multitouch.Configuration.WPF.exe. Lorsque cet utilitaire aura démarré, cliquez sur le bouton 'Configure Device', puis dans le dialogue qui apparaît, cochez la seule case présente, afin de faire disparaître le pointeur natif, comme illustré [Fig.3]. Tout est prêt. Se promènent alors sur votre écran, selon les actions sur les souris, autant de points rouges qui sont autant de traces de doigts :-) Pour stopper l'émulateur, il suffit de presser la touche [ENTER] dans la fenêtre du driver, puis dans la fenêtre du service précédemment lancé.

4 PREMIERS PAS

Une application qui veut supporter le tactile doit d'abord s'enquérir de la disponibilité de celui-ci. Nous renvoyons le lecteur à l'exemple DeviceTest sur notre site. Attention, tout programme tactile doit être compilé pour Windows 7, ce qui revient à définir une macro ainsi :

```
#define WINVER 0x0700
```

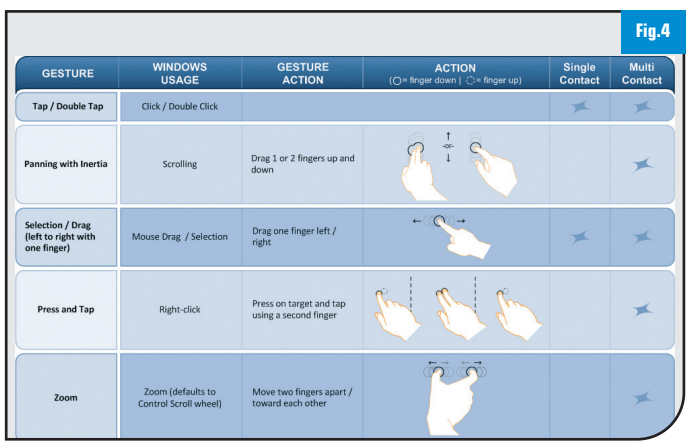
Nous renvoyons le lecteur au fichier targetver.h des exemples sur notre site.

5 PREMIER ESSAI AVEC LES GESTURES

Ce que la documentation du SDK Windows 7 appelle gestures [Fig.4] est un ensemble de déplacements conventionnels et bien définis d'un ou deux doigts, et que le système interprète, par exemple, un scrolling, un double clic, un mouvement de rotation, etc. Une application est notifiée d'une gesture effectuée par l'utilisateur via un message WM_GESTURE. Voyons de quoi il s'agit. Créez une application Win32 avec Visual Studio, ou reprenez le code de notre exemple BasicGesture, disponible sur notre site. Comme vous le constatez, le code d'une application Windows brute est plutôt verbeux. Le cœur d'une telle application est comme ceci :

```
while (GetMessage(&msg, NULL, 0, 0))
{
  if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
}
```

Fig.4

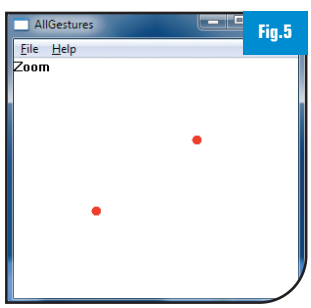


Les gestures, telles que définies par la documentation Microsoft.

Cette boucle est une pompe à messages qui se charge d'extraire d'une file les messages à destination de l'application. Quand un message est reçu, il est passé par un mécanisme interne de Windows à la procédure de fenêtre (WndProc), qui est une fonction de rappel invoquée pour l'occasion. Dans cette procédure, le programmeur doit tester de quel message il s'agit, en principe au sein d'une construction switch-case vertigineuse et parfaitement rebutante. En voici un extrait, tiré de l'exemple BasicGesture :

```
switch (message)
{

case WM_PAINT:
  hdc = BeginPaint(hWnd, &ps);
  //etc
  EndPaint(hWnd, &ps);
  break;
case WM_GESTURE:
  return HandleGesture(hWnd, message, wParam, lParam);
case WM_DESTROY:
  PostQuitMessage(0);
  break;
default:
  return DefWindowProc(hWnd, message, wParam, lParam);
}
```

Fig.5



Une gesture zoom, reconnue par notre programme de démonstration

Dans notre exemple, quand le message WM_GESTURE est rencontré nous le passons à une fonction maison: HandleGesture, dont le principe est simple, le paramètre lParam du message est un Handle qui permet de remplir une structure GESTUREINFO via l'API GetGestureInfo. L'analyse du contenu de la structure GESTUREINFO permet de savoir de quelle gesture il s'agit [Fig.5]. En programmation Win32, le traitement d'un message par la procédure de fenêtre doit être signalé en retournant la valeur zéro, ou, sinon, on laisse l'API DefWindowProc s'occuper de cette question. En outre ici, la documentation précise que le programmeur doit appeler API CloseGestureInfoHandle, faute de quoi, il y a fuite de ressources. Toutefois, les extraits de code de la documentation ne le font pas toujours très rigoureusement. Restons donc vigilants.

6 TOUTES LES GESTURES

Si vous avez essayé notre premier exemple, vous avez eu la surprise de constater que par défaut, et pour une raison inexplicable, le système ne reconnaît pas toutes les gestures. La gesture rotation est manquante notamment. Windows 7 permet de sélectionner quelle(s) gesture(s) seront transmises à l'application. Cette configuration qui se fait via l'API SetGestureConfig, peut être faite une fois

pour toutes, par exemple au démarrage de l'application, ou bien à la volée. Si une gesture est signalée par l'application, il existera un message qui signale qu'une gesture va être signalée :-) Ce message est WM_GESTURENOTIFY. C'est lors du traitement de ce message qu'il est possible de spécifier quelles gestures devront être ultérieurement reçues. Pour cela, on remplit un tableau de structures GESTURECONFIG. Voici un extrait de notre exemple AllGestures, qui active toutes les gestures :

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
 LPARAM lParam)
{
 int wmId, wmEvent;
 PAINTSTRUCT ps;
 HDC hdc;

 switch (message)
 {
 case WM_GESTURENOTIFY:
 {
   GESTURECONFIG gc =
   {
     0,// global gesture configuration flags
     GC_ALLGESTURES, // Toutes les gestures activées
     0 // aucune gesture désactivée
   };

   SetGestureConfig(
     hWnd,
     0,
     1,// Une seule structure passée à
       //l'API dans le tableau ci-dessous
     &gc,// Tableau de structures GESTURECONFIG
                 sizeof(GESTURECONFIG));
 }
 break;

 // etc
}
```

La structure GESTURECONFIG contient trois membres. Le premier est un identifiant de la gesture que l'on veut configurer. Les deuxième et troisième membres permettent d'affiner cette configuration. Ainsi, si l'on donne GID_PAN (Gesture Scrolling) dans le premier membre, et que l'on donne GC_PAN_WITH_SINGLE_FINGER_VERTICALLY en second membre et GC_PAN_WITH_SINGLE_FINGER_HORIZONTALLY en troisième membre, le système reconnaîtra seulement comme valable la gesture qui produit un scrolling vertical avec un seul doigt. Dans notre exemple nous utilisons la possibilité d'activer toutes les gestures d'un seul coup. Ainsi la rotation n'est plus manquante. D'après la documentation, le tableau que l'on passe à l'API SetGestureConfig n'est pas limité en taille, et les structures de configuration sont interprétées dans l'ordre selon lequel elles figurent dans le tableau. Si des structures de configuration se contredisent, par exemple en activant puis en désactivant une gesture particulière, c'est la dernière structure interprétée qui prime. Cependant nos essais montrent que la pratique ne rejoint pas toujours la théorie. Ainsi l'on pourrait penser que si l'on désactive toutes les gestures d'un coup comme ceci :

```
GESTURECONFIG gc =
{
 0,// global gesture configuration flags
 0,// aucune gesture activée
 GC_ALLGESTURES // toutes les gestures désactivées
};
```

Puis que l'on en active une seule, on aura celle-ci seulement activée. Sauf erreur de votre serviteur, cela ne fonctionne pas bien et on peut se retrouver sans gesture activée du tout.

7 ACTIVER UNE SEULE GESTURE, CAS DE LA ROTATION

Comment faire alors pour qu'une seule gesture soit activée, la rotation par exemple ? La solution consistant à traiter deux messages WM_GESTURENOTIFY à la suite est malhabile. Tout simplement, on commencera par tout désactiver au démarrage de l'application. Dans notre exemple, RotationGesture disponilbe sur notre site, l'endroit est la fonction InitInstance, dont voici un extrait du code :

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

 //etc

 GESTURECONFIG gc[] =
 {
   {
     0, // global gesture configuration flags
     0, // aucune gesture activée
     GC_ALLGESTURES, // Toutes les gestures désactivées
   },
 };

 SetGestureConfig(
   hWnd,
   0,
   sizeof(gc)/sizeof(GESTURECONFIG),
   (PGESTURECONFIG)&gc,
   sizeof(GESTURECONFIG));

 // etc
}
```

Puis on active ce que l'on veut en réponse au premier message WM_GESTURENOTIFY :

```
case WM_GESTURENOTIFY:
{
 GESTURECONFIG gc[] =
 {
   {
     GID_ROTATE, // On configure la rotation
     GC_ROTATE, // ROTATION demandée
     0, // Rien n'est désactivé sour GID_ROTATE
   },
 };

 SetGestureConfig(hWnd, 0,
   sizeof(gc)/sizeof(GESTURECONFIG),
```

```
    (PGESTURECONFIG)&gc?
    sizeof(GESTURECONFIG));
}
break;
```

Bien lire ce que dit la documentation à propos des données fournies par le système pour une gesture. Ces données sont fournies brutes et doivent être traitées. Ainsi dans le cas de la rotation, l'angle de celle-ci est codé sur 16 bits, le bit de poids fort servant à déterminer le sens de la rotation. L'angle est exprimé en radian. Le code partiel ci-dessous affiche la valeur de cet angle en degrés, au fur et à mesure des messages reçus.

```
#include <sstream>
using namespace std;

void WriteRotation(HWND hWnd, USHORT arg)
{
 wostringstream wss;
 wss << L"Rotation, angle: ";
 double alpha GID_ROTATE_ANGLE_FROM_ARGUMENT(arg);
 alpha = alpha*180.0/3.14159265359;
 wss << alpha;

 HDC hdc = GetDC(hWnd);
 SelectObject(hdc, GetStockObject(WHITE_PEN));
 Rectangle(hdc, 0, 0, 200, 20);
 TextOut(hdc, 0 , 0,
   wss.str().c_str(), wss.str().length());
 ReleaseDC(hWnd, hdc);
}


GESTUREINFO gi;
ZeroMemory(&gi, sizeof(GESTUREINFO));
gi.cbSize = sizeof(GESTUREINFO);

BOOL result  = GetGestureInfo((HGESTUREINFO)lParam, &gi);

case GID_ROTATE:
 WriteRotation(hWnd, (USHORT)gi.ullArguments);
break;
```

La valeur de l'angle est fournie de façon incrémentale à chaque message, Par exemple 50°, 51 °, 52°, etc. Cette valeur est donc l'angle à l'instant t dans le mouvement. Mais si l'on veut dessiner une figure qui soit animée en suivant la gesture faite par l'utilisateur, cette valeur ne convient pas. Ce qu'il faut est le pas de rotation que le programmeur doit calculer en retranchant à l'angle à l'instant t, l'angle de l'instant précédent qu'il aura pris soin de conserver. Chaque gesture a ainsi ses particularités. Ainsi, pour prendre un autre exemple, la position d'un clic (gesture Tap) est exprimée en coordonnées écran par le système. Il incombe au programmeur d'appeler l'API ScreenToClient pour connaître la position du clic dans la fenêtre de son application.

8 L'API TOUCH

Contrairement aux gestures, avec l'API Touch il est possible de travailler en multipoints. Jusqu'à 255 points avec l'émulateur que nous utilisons. On enclenche l'API Touch au démarrage de l'application, comme ceci :

```
RegisterTouchWindow(hWnd, 0);
```

C'est tout simple, et même tellement simple qu'à partir de ce moment les gestures sont irrémédiablement désactivées. Plus de message WM_GESTURE, mais un message WM_TOUCH en remplacement. L'API Touch se contente de fournir un tableau de points à chaque événement. La fonction OnTouch de notre exemple BasicTouch illustre comment traiter ces données.

```
LRESULT OnTouch(HWND hWnd, WPARAM wParam, LPARAM lParam )
{
    UINT cInputs = LOWORD(wParam);
 if(!cInputs)
   return DefWindowProc(hWnd, WM_TOUCH, wParam, lParam);

 int x = 0;
 int y = 0;
 HDC hdc = GetDC(hWnd);
 TEXTMETRIC tm;
 GetTextMetrics(hdc, &tm);
 int h = (int)(tm.tmHeight + tm.tmExternalLeading);
 SelectObject(hdc, GetStockObject(WHITE_PEN));
 Rectangle(hdc, 0, 0, 500, 40);

 PTOUCHINPUT pInputs = new TOUCHINPUT[cInputs];
 GetTouchInputInfo((HTOUCHINPUT)lParam, cInputs,
   pInputs, sizeof(TOUCHINPUT));
 for (UINT i=0; i < cInputs; i++)
 {
    TOUCHINPUT ti = pInputs[i];
    PrintTouch(hdc, x, y, &ti);
    y += h;
   }

   delete [] pInputs;
 ReleaseDC(hWnd, hdc);
 return 0;
}

case WM_TOUCH:
 OnTouch(hWnd, wParam, lParam);
 break;
```

Toutefois, Windows 7 fournit deux interfaces COM pour aider le programmeur dans le traitement des données brutes fournies via WM_TOUCH. Ce sont les interfaces COM IManipulationProcessor et IInertiaProcessor, que nous découvrirons le mois prochain.

9 CONCLUSION PROVISOIRE

Nous avons vu les bases de la prise en charge des périphériques tactiles. Nous avons constaté qu'en Win32, le code est vite lourd. On gagnera à utiliser un toolkit graphique, pourvu que celui-ci permette de manipuler la table de routage des messages afin de réceptionner WM_GESTURE ou WM_TOUCH. À bientôt.

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

WEB

Plus loin avec symfony 1.3 et 1.4

*Difficulté : *****
Éditeur : Sensio Labs books
Auteur : Fabien Potencier
Prix : 29,90 €

Vous rêvez d'être un maître ès symfony ? « Plus loin avec symfony » est fait pour vous ! Il est la référence ultime pour les fonctions avancées, les manipulations expertes au cœur même du framework. On ne débutera pas symfony avec cet ouvrage. Si vous avez déjà voulu savoir comment il fonctionne à l'intérieur, ou bien si vous souhaitez étendre le framework de diverses manières en vue de le faire fonctionner pour des besoins spécifiques, alors ce livre est fait pour vous. Dans ce cas précis, il contient tout ce qu'il faut savoir pour faire évoluer vos compétences symfony au niveau supérieur. Un must !

LANGAGE

Objective-C pour le développeur avancé

*Difficulté : *****
Éditeur : Eyrolles
Auteur : Pierre Y. Chatelier
Prix : 32 €

Langage méconnu et assez marginal jusqu'à la déferlante iPhone/iPad, Objective-C hérite de l'expérience du C. Il est le langage historique de NeXT et désormais de MacOS X et de tout l'environnement Apple. On se surprend toujours devant sa richesse insoupçonnée. Et l'auteur nous en livre ici un exemple fonctionnel, surtout depuis la disponibilité de la version 2 du langage. Pas toujours simple d'accès, quand on le maîtrise, on peut alors réaliser des choses étonnantes à l'égal d'un C++. L'ouvrage aborde l'ensemble des éléments avancés du langage pour gagner en productivité, en optimisation.

STANDARD

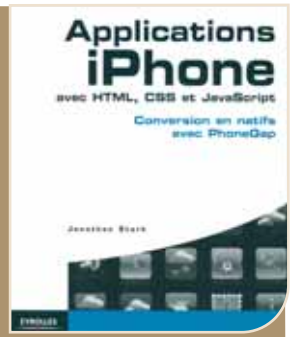
Bonnes pratiques des standards du web

*Difficulté : ***
Éditeur : Pearson
Auteur : Dan Cederholm
Prix : 28 €

Il existe pléthore de standards du web ou prétendus standards. Mais il n'est pas toujours simple de faire un tri, de les comprendre et de les utiliser dans ses développements web. Les chapitres équilibrent aspects théoriques et utilisation pratique des concepts présentés, et leur approche modulaire permet d'assimiler, comprendre et exploiter rapidement l'essentiel des standards du Web. Chaque aspect fait l'objet d'une discussion et présente différentes solutions. L'auteur aborde de nombreux exemples facilitant la compréhension.

LIVRE DU MOIS

Applications iPhone



*Difficulté : *** - Éditeur : Eyrolles*
Auteur : Jonathan Stark - Prix : 20 €

Dans ce nouvel ouvrage dédié à l'iPhone, l'auteur décide de se passer de Xcode et surtout d'Objective-C pour utiliser les standards du web qui permettent de construire des applications pour le téléphone pommé : html, CSS et Javascript. Le tout en passant par le framework open source, PhoneGap. On débute par les bases du développement web pour mobile et iPhone avec les bonnes pratiques, les rudiments à acquérir. Mais dans les faits, il s'agit de réellement créer des applications natives via les outils du monde web. Cela implique une bonne dose de CSS et de Javascript. Avantage : c'est plus léger et on peut faire des choses très intéressantes (regardez les démos autour de HTML 5). PhoneGap permet quant à lui de générer une application native depuis son projet « web ». Mais dans ce cas, vous devez passer par le Mac et Xcode. Eh oui, on n'échappe pas à son destin iPhone. À vous de tester !

WEB

Le référencement sur Google

*Difficulté : ***
Éditeur : Micro Application
Auteur : Gilles Grégoire
Prix : 16,50 €

Incontournable, le moteur de recherche Google s'impose aux webmasters pour optimiser les options de référencement. Un travail fastidieux qu'il faut tout le temps vérifier, refaire, car les choses vont vite sur internet. Ce guide complet explique l'ensemble des techniques, des options, des règles pour avoir un bon référencement. Et la liste est très longue : liens, page rank, netlinking, description, mots-clés, etc.

WEB

WinDev 15

*Difficulté : ****
Éditeur : éditions Eni
Auteur David Vandevelde
Prix : 29,90 €

Tout en un du développement et de la gestion projet, WinDev 15 regroupe des centaines de fonctions, des modules UML, de déploiement, de tests, pour les terminaux mobiles, pour le web et supporte de nombreux langages en plus de son langage maison : WLangage. L'auteur, bon expert de l'outil, décrit d'abord l'ergonomie générale pour la définition des données, des fenêtres, des impressions, l'écriture du code, l'aide en ligne. Il présente ensuite de façon exhaustive les objets utilisables dans les fenêtres et dans les états. Il expose les éléments du WLangage et le principe du RAD... Un bon ouvrage pour débuter avec WinDev...

SYSTÈME

Windows 7 déploiement et administration

*Difficulté : ****
Éditeur : Dunod
Auteur : collectif
Prix : 45 €

Destiné avant tout à celles et ceux qui doivent déployer et administrer Windows 7, ce livre est une mine d'or ! Comment migrer un parc sans trop de douleur ? Comment s'administrent des machines Windows 7 ? Les auteurs abordent l'ensemble des problématiques : WAIK, WDS, deux outils indispensables.
L'approche infrastructure n'est pas oubliée, pour avoir une structure adaptée et la plus performante possible.

DÉVELOPPEZ 10 FOIS PLUS VITE

WINDEV®

ENVIRONNEMENT PROFESSIONNEL DE DÉVELOPPEMENT (AGL)

Créez des applications pour Windows, Internet, Mobile et SaaS
Java, .Net, PHP, J2EE, Webservice, XML, Ajax, Linux, Android, Client riche, Base de Données...

AGL N°1 EN FRANCE

Demandez le dossier + DVD gratuit

Elu «**Langage le plus productif du marché**»

Tout est inclus
Gère le cycle complet de développement.
Support Technique gratuit

555 NOUVEAUTÉS

VERSION EXPRESS GRATUITE
Téléchargez-la !

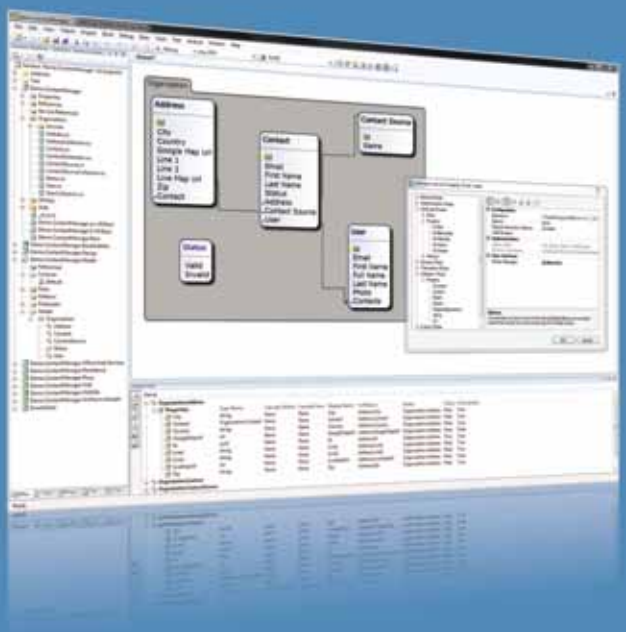
PC SOFT 25ème ANNÉE Au service des développeurs

Fournisseur Officiel de la Préparation Olympique

www.pcsoft.fr

Dossier gratuit 200 pages sur simple demande. Tél: **04.67.032.032** info@pcsoft.fr

Besoin D'AIDE ?
CODEFLUENT ENTITIES



ELÉMENTS APPLICATIFS

- *Localisation*
- *Liaison aux données*
- *Règles & Validation*
- *Concurrence d'accès*
- *Sécurité*
- *Cache*
- *Gestion des BLOBS*

ARCHITECTURES

- *SOA, Client intelligent*
- *Client riche, RIA*
- *Web, Webparts*
- *Client/Serveur, N-Tier*
- *Office*
- *SharePoint*
- *SaaS, Cloud*

TECHNOLOGIES

- *.NET, C#, Linq, Visual Studio*
- *ASP .NET WebForms, MVC*
- *Silverlight, WCF, ASMX*
- *WPF, WinForms*
- *Microsoft SQL Server*
- *Oracle Database*
- *Excel, Access, SharePoint*

LA PREMIERE FABRIQUE LOGICIELLE PLEBISCITEE PAR LES CLIENTS

Face aux enjeux associés à l'**évolution constante des technologies** et à leur intégration, CodeFluent Entities, la fabrique logicielle pilotée par les modèles apporte une solution concrète immédiate.

CodeFluent Entities offre aux architectes et aux développeurs **une méthode structurée** et tous les **outils** pour faciliter le développement des applications .NET, sur tout type d'architecture, à partir d'une **modélisation du métier.**
CodeFluent Entities permet de réaliser des applications **évolutives** de **qualité** avec un niveau de **productivité** inégalé.

*Téléchargez votre licence **GRATUITE** ici :*
http://www.CodeFluentEntities.com/programmez

CodeFluent Entities
the Model Driven Software factOry

Contact: info@softfluent.com
www.CodeFluentEntities.com
***SoftFluent** 3 rue de la Renaissance 92160 ANTONY*