PROGRAMMEZ !
mensuel n°115 - janvier 2009
www.programmez.com

PROgrammez !

Nouvelle Formule

Le magazine du développement

Boostez Java !

Doper Java 7 au multicore
Astuces pour optimiser votre code Java
Les architectures Java EE

CD-Rom virtuel
réservé aux lecteurs

Logiciels à télécharger
en AVANT-PREMIÈRE

CARRIÈRE
Développeur FREELANCE

La MODÉLISATION de A à Z

Quels modèles et outils choisir ?
Mes premiers modèles en quelques clics
Les bonnes pratiques UML

Linux

Voyage au cœur du noyau Linux

Vidéo SUR LE WEB

Coder les API YouTube !
Vidéo live en streaming sur votre site

.Net
- **Développer une application Speech Server**
- **Pie Menu : un menu original à découvrir !**

Google
- **Mettez du drag & drop dans GWT**

Architecture
- **Les atouts de OSGi**

Langage
- **Découvrez Rebol !**

M 04319 - 115 - F: 5,95 €

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

PROgrammez ! le magazine du développement

numéro 115 - janvier 2009

sommaire

\\ actus

L'actualité en bref 6
Agenda 8

\\ événement

Au cœur du noyau Linux 10

\\ dossier : boostez java !

Java 7 Fork / Join : exploiter toute la puissance disponible 17
Paramétrage des EJB3 en XML 20
Invocation de langages dynamiques en Java 22
Les bonnes pratiques d'optimisation du langage Java 25
Java EE 5 : pour vos architectures 27

\\ projet : la modélisation de A à Z

Le challenge de la modélisation pour le développeur 31
Mes premiers pas en modélisation UML 32
Générer du code avec UML en quelques clics 36
La modélisation au cœur des applications métier 38
Quel modèle pour quel usage ? 39
MDSD, MDD, ou MDA : quel outil choisir ? 42
Modélisation intégrée : la mettre en œuvre 45
Utiliser ou non UML, telle est la question 46
Panorama des outils 48

\\ développement web

Dopez votre site web à la vidéo

Silverlight Streaming et Silverlight 2 : tout par la vidéo 51
Développer avec les API YouTube 53
Faites de la vidéo en live sur votre site web ! 56

\\ carrière

Epitech, le réseau et la passion 58
Travailler en freelance 60

\\ code

Un menu original : le Pie Menu ! 62
Développer une application Speech Server avec le framework.net 65
L'architecture et la technologie OSGI 68
Programmez OpenGL avec le langage Ruby (2e partie) 71
A la découverte de Rebol 75
Réaliser un composant " glisser-déposer " avec GWT 79

\\ temps libre

Les livres du mois 82

CD ROM virtuel

Réservé aux lecteurs : téléchargement sur www.programmez.com

Les outils du mois !

Amethyst PE : un IDE Flex complet
Développer en Flex sous Visual Studio ? C'est désormais possible avec Amethyst, un IDE basé sur Visual Studio Shell !

Ilog Jviews Diagrammer 8.5
Nouvelle version de la suite de visualisation Java d'ILOG avec toujours plus de RIA, d'Ajax, de composants ! Windows

Red5 v0.8.0 rc
Déployez rapidement votre serveur Flash pour vos vidéos et animations grâce à Red5, un serveur open source.

Spécial modélisation

Objecteering Next UML Free Edition
Outil de modélisation UML. Objecteering "NEXT" Free Edition vous donne accès en mode mono-utilisateur à toutes les fonctionnalités standard du modeleur.

mia-studio 6.0.3
Solutions pour améliorer l'agilité dans le développement et la maintenance logiciels.

Lyria Leonardi Free 4.0
Framework MDA en Java / XML permettant d'automatiser la production d'interfaces homme-machine.

Java / RIA

Javafx 1.0 SDK
La toute nouvelle plate-forme de développement d'application internet riche (RIA) en Java ! A découvrir dès aujourd'hui !

Netbeans 6.5 Java FX
Découvrez la toute dernière version de l'IDE Java open source NetBeans+ Java FX.

PHP

Symfony sandbox
Installez dès aujourd'hui la sandbox du Framework PHP Symfony. Un package complet pour développer plus rapidement en PHP !

LEONARDi

W4-LYRIA : la modélisation au coeur des applications métier

L'ihm

EN TOUTE SIMPLICITÉ !

Pour implémenter des applications riches, accédant à des données hétérogènes (applicatifs métier, BD, et depuis la V4.1, Hibernate, GED et silos documentaires), déployées en Swing, plugin Eclipse ou DHTML/Ajax et s'insérant naturellement dans les processus d'entreprise...

Adoptez la puissance et l'agilité de l'approche Model-Driven

NOUVELLE VERSION LEONARDI V4.1 open source

Concentrez-vous sur votre métier et dotez votre entreprise d'avantages compétitifs durables: amélioration du cycle de vie du logiciel, démarche itérative par prototypage pour coller aux besoins, découplage technologie/métier, évolutivité, agilité et évolutivité, le tout sans expertise technique pointue !

LYRIA

Pour en savoir plus sur la solution Leonardi, rendez-vous sur notre site www.lyria.com ou envoyez-nous un courriel à info@lyria.com

éditoll

Vous croyez encore au Père Noël ? Nous aussi !

On ne peut pas dire que la crise, irrationnelle dans son comportement, laisse beaucoup de monde indifférent. Et si au départ, certains ont cru que le " High tech " serait à l'abri, ce n'est plus réellement le cas. Mais la crise peut aussi avoir du bon...

" A qui profite la crise ? " A l'open source ? S'il n'y a pas de licences à payer, les services restent indispensables et donc les clients en consomment. Et le DSI peut dire à son responsable financier : " j'économise même si je dépense ". L'équation parfaite ! De là à conclure que l'open source sortira grand vainqueur paraît prématuré. Même si sans doute les logiciels ouverts peuvent effectivement prendre une place plus importante dans la bureautique, les outils de commodité, et désormais le progiciel. La crise n'annulera pas les projets stratégiques ou critiques, là où les outils du commerce gardent toujours une bonne place.

On parle depuis des années d'usines à logiciels, de modélisation, de bonnes pratiques. La crise peut justement remettre la question sur les claviers. Si une des préoccupations est de savoir comment rendre plus efficaces les équipes de développeurs, la modélisation est une des réponses. Et elle aura un impact de plus en plus important sur le développeur, qu'il le veuille ou non. Notre dossier du mois vous le prouvera.

Toute l'équipe de Programmez vous souhaite une bonne année 2009, avec plein de code, plein d'IDE et tout un tas de technologies qui nous lasseront plus ou moins vite !

François **Tonic**
Rédacteur en chef

Le CD-ROM de PROGRAMMEZ! devient virtuel

C'est un changement important. Depuis son N°1, votre magazine était accompagné d'un CD-Rom. Ce 115e numéro inaugure un nouveau modèle, permettant au lecteur de télécharger les logiciels sélectionnés sur le site programmez .com. On se posait la question depuis plus de deux ans : avec la généralisation du haut débit, le cd-rom s'impose-t-il encore ?
Car bien entendu, il représente un budget de fabrication important, alors qu'en dix ans, le prix de vente du magazine était resté identique, malgré l'inflation des coûts ! Désormais, le cd-rom ne sera plus systématique. Mais remplacé par le " CD virtuel ", en téléchargement, ou bien réalisé, en partenariat avec des éditeurs.

Mais cette nouvelle ère peut recéler des avantages. La dématérialisation dispense des limites et des contraintes physiques : il n'y plus de problème de taille du cd-rom !
Et nous travaillons pour vous faire connaître une autre forme d'expérience, au travers du " CD-rom virtuel ".
Celui-ci permet davantage d'interactivité, avec une interface qui se sophistiquera au fur et à mesure des numéros. Et de réactivité : la dernière version des logiciels pourra même se substituer aux versions précédentes.
Enfin, avec les éditeurs, nous aurons comme objectif de vous réserver chaque mois des logiciels en exclusivité, ou en avant-première.

Jean **Kaminsky**
Directeur de la rédaction

PROgrammez! Le magazine du développement

Rédaction : redaction@programmez.com
Directeur de la Rédaction : Jean Kaminsky
Rédacteur en Chef : François Tonic
Ont collaboré : F. Mazué, C.Remy.
Experts : J. Delvare, D. Martin, D. Mera, S. Saurel, A. Goncalves, F. Thomas, J-L. Comeliau, P. Desfray, A. Buisine, C. Vidal, F. Barbier, T. Matusiak, G. André, L. Bar, C. Lomba, D. Négrier, R. Tomczak, E. Vernié, D. Caro, T. Templier, H. Darmet, X. Blanc.
Crédits photo : DR - Maquette : AJE Conseils
Publicité : Régie publicitaire, K-Now sarl
Pour la publicité uniquement :
Tél. : 01 41 77 16 03 - diff@programmez.com
Editeur : Go-02 sarl, 6 rue Bezout
75014 Paris - diff@programmez.com
Dépôt légal : à parution - Commission paritaire : 0712K78366 ISSN : 1627-0908
Imprimeur : ETC - 76198 Yvetot
Directeur de la publication : J-C Vaudecrane

Abonnement : Programmez 22, rue René Boulanger, 75472 Paris Cedex 10 - abonnements .programmez@groupe-gli.com Tél. : 01 55 56 70 55 - Fax : 01 55 56 70 20 - 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 : 45 € - Etudiant : 39 € - CEE et Suisse : 51,83 € Algérie, Maroc, Tunisie : 55,95 € Canada : 64,33 € Tom : 79,61 € - Dom : 62,84 € Autres pays : nous consulter. **PDF :** 30 € (Monde Entier) souscription en ligne.

L'INFO PERMANENTE
WWW.PROGRAMMEZ.COM

PROCHAIN NUMÉRO
N°116 février 2009, parution 31 janvier

✔ **Développement orienté**
COMPOSANTS
Réduire le code à écrire, ne pas réinventer la roue à chaque projet !
Savoir choisir et utiliser les composants.

✔ **SGBD**
Bases de données :
optimisations, bonnes pratiques

✔ **Développement Web**
Devenez un maître en **ASP.Net**

■ **PHP 5.3** sera disponible dans quelques semaines et intégrera bien des nouveautés, dont beaucoup étaient prévues dans la v6, comme les espaces de noms ou l'internationalisation accrue et une meilleure gestion des dates. On attend toujours une roadmap et une liste de fonctions pour la v6.

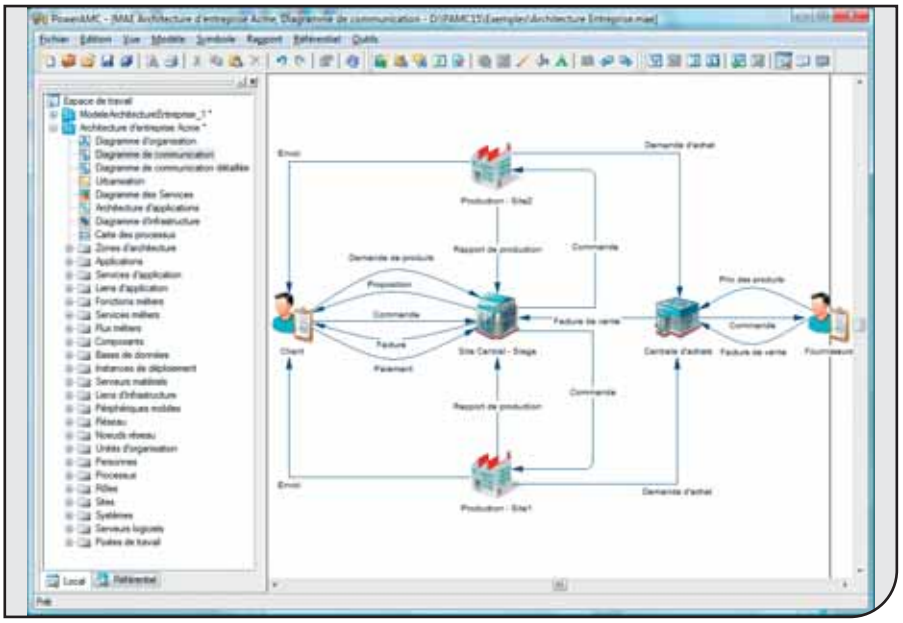
■ **Ingres** a lancé fin novembre la v9.2 de Ingres Database. Cette version vise à réduire l'administration, notamment en automatisant toujours plus de tâches. Le SGBD renforce aussi la prise en charge d'Unicode, offre une disponibilité serveur plus élevée. Côté API, cette version fait évoluer le support des librairies JDBC, ODBC, .Net.

■ **JetBrains** lance la 4e version de TeamCity. Il s'agit d'un outil d'intégration continue et d'une gestion de build. Il permet d'automatiser les tâches et d'améliorer le cycle de développement, notamment pour les équipes de développeurs. Très ouvert, il est facilement extensible avec une API. Cette édition améliore l'intégration à Eclipse.

■ L'éditeur français **4D** lance la version Release 3 de son serveur de données 4D Server. Cette mouture prend mieux en compte les machines multicoeurs, avec une grosse optimisation du multithread. On bénéficie aussi d'un nouveau protocole de connexion SQL évitant de passer par un pilote ODB quand on fait collaborer deux applications 4D. Sur SQL, notons le support des schémas SQL. Dans les échanges, l'éditeur a rajouté un nouveau mécanisme de compression pour les services SOAP. SVG est désormais pleinement supporté.

■ **Quest Software** propose maintenant de connecter son outil JProbe à Eclipse ! Ainsi JProbe devient un plug-in de son environnement de développement. Rappelons que JProbe est un outil de profiling des applications Java. Le plug-in reprend les 4 grandes fonctions : diagnostic des performances, analyse d'utilisation de la mémoire, optimisation des performances et taux de couverture de tests unitaires.

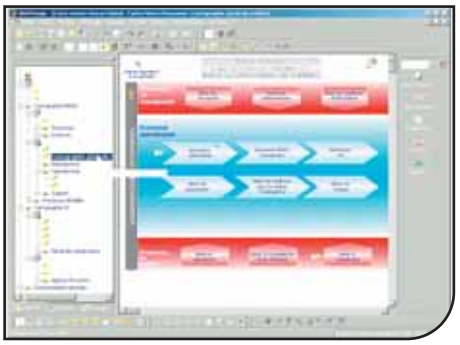
Modélisation
Sybase lance PowerAMC 15



Depuis AMC Designor, l'outil de modélisation PowerAMC a bien changé même si le concept de base reste identique : modéliser les architectures d'entreprise, les processus IT. Cette nouvelle version majeure améliore de nombreux éléments, comme la présence d'une liaison – synchronisation capable d'identifier les intersections entre les domaines, une nouvelle manière de capturer et de collecter les métadonnées pour l'architecture d'entreprise. On peut désormais partager le référentiel entre tous les acteurs via une nouvelle interface web. On peut aussi reprendre les métadonnées métiers issues de Microsoft Visio. Notons que la partie projet a été profondément revue avec une meilleure expérience pour travailler sur plusieurs modèles et les fichiers interconnectés. Et on dispose aussi de Template de projet.
La partie données a été améliorée avec le support de la notation Barker, d'un nouveau module Modèle logique de données, en fournissant une abstraction entre le modèle conception de données et le modèle physique de données. L'outil supporte aussi les dernières versions des SGBD, même si concernant SQL Server 2008 ce support reste limité. Dans la partie fluidité de l'information, Sybase inclut de nouveaux assistants ETL. Notons aussi le support de EMF (Eclipse) ou encore la présence d'un nouveau modèle d'analyse d'impact ! Bref, une version importante qui satisfera de nombreux utilisateurs anciens et nouveaux.
Site : www. Sybase.com

Modélisation
Win Design bientôt en version 9

Prévue pour courant janvier 2009, la v9 du modeleur Win Design (édité par Cecima) comprendra deux grosses nouveautés : une extension des capacités de paramétrage du méta modèle et un documenteur. Celui de la v9 facilitera l'extraction et la présentation des objets modélisés pour l'éditeur des dossiers papiers du projet. Sur les paramétrages, il s'agit d'ouvrir toujours plus le méta modèle pour le personnaliser, rajouter des éléments, les modifier, etc.



DES EXPÉRIENCES INTERFACES UTILISATEUR QUI INSPIRENT

Introducing...

NetAdvantage® .NET ASP.NET, WinForms, WPF, Silverlight

NetAdvantage® win client Windows Forms, WPF

NetAdvantage® web client ASP.NET, Silverlight

NetAdvantage® Select ASP.NET, Windows Forms

NetAdvantage® Data Visualization Silverlight

NetAdvantage® ICONS Office Basics

Pour vos applications .NET actuelles… et futures

En tant que développeur, votre défi est de concevoir des applications en utilisant les plates-formes les plus matures et robustes disponibles maintenant, Windows Forms ou ASP.NET, tout en planifiant les nouvelles technologies émergentes telles que WPF et Silverlight™. C'est la raison pour laquelle nous alignons nos produits selon vos besoins.

Dans un monde en constant changement, vous pouvez toujours faire confiance aux valeurs durables de NetAdvantage : la puissance et la qualité de ses composants UI ; la documentation efficace ; les tutoriaux instructifs ; l'assistance technique fiable ; et des services de consultation récompensés afin de permettre aux développeurs de livrer à leurs utilisateurs une meilleure expérience dans vos applications d'aujourd'hui et celles de demain.

Pout tout renseignement:
infragistics.com/thefutureisnow

Appelez dès aujourd'hui

N° Vert 0800 667 307

Infragistics®

■ **Nuxeo,** éditeur français de solutions de gestion de contenu, poursuit son développement après une réorganisation interne de son fonctionnement. Deux axes sont à noter pour les prochains mois : le développement de solutions packagées orientées métiers, l'intégration du protocole Sharepoint de Microsoft et l'ouverture de bureaux à l'étranger. L'internationalisation est une des priorités pour 2009 (USA : 1er trimestre, Allemagne : 3e trimestre).

■ **NetBeans 6.5** est disponible en version finale depuis fin novembre dernier. Le développement web est une des crédos de cette version avec une intégration toujours poussée de PHP, Ruby, Rails. Autre grosse nouveauté le support par défaut de JavaFX 1.0 !

■ **Forum PHP 2008** a connu un gros succès avec plus de 250 personnes par jour ! Plusieurs cas clients importants (BNP, 20 Minutes) présentaient leur projet PHP. Parmi les exposants, nous pouvions noter la présence d'un stand Oracle ! L'éditeur veut mieux s'insérer dans le monde PHP, car les clients le demandent. Oracle travaille déjà avec Zend mais cela ne devrait pas s'arrêter là !

■ Un patch pour **MySQL 5.1** est en développement chez l'éditeur. Une polémique avait eu lieu lors de la sortie de la v5.1 à cause de bugs supposés. L'affaire semble aujourd'hui à peu près éteinte.

■ **Le W3C** s'intéresse activement aux technologies mobiles et à leur impact sur le développement et le potentiel d'innovation dans les pays émergents. *" Outre sa capacité à garantir l'accès Internet, la pénétration et le faible coût de la téléphonie mobile dans les pays en développement nous donnent la possibilité d'offrir des services Internet à des milliards de personnes qui ne pourraient pas par ailleurs en bénéficier "* explique George Sadowsky, co-responsable de l'atelier.

■ **Embarcadero** a dévoilé début décembre RAD Studio 2009 pour les développements .net et mono. Ce package inclut Delphi, C++ Builder et le tout nouvel environnement de développement Prism pour la plate-forme .Net. D'autre part, l'éditeur annonçait aussi ER / Studio 8, outil de modélisation de données. Parmi les nouveautés : support LDAP, des derniers SGBD.

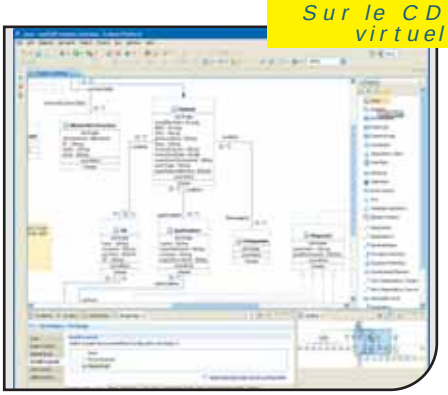
Technologie
Google se met aussi au RIA

Décidément on n'arrête pas Google. Aujourd'hui, l'éditeur s'attaque aux plates-formes RIA avec sa propre technologie : **Native Client.** Jusqu'à présent, il fallait faire son choix entre Adobe et Microsoft principalement. Sun tente de réagir avec JavaFX disponible en v1 mais son accueil par le marché est pour le moins tiède et il doit démontrer sa réelle utilité ! Native Client est une technologie web indépendante des OS et des navigateurs. Mais il est capable d'intégrer des modules natifs. Le cœur de Native Client repose sur un plug-in servant de " core ". Et chaque module doit être validé. Les outils de compilation reposent sur gcc. Pour Google, le challenge est grand, car la concurrence est vive et il faudra prouver la pertinence de la technologie et surtout convaincre les développeurs. Il s'agit de pouvoir créer des applications riches comme avec Flex ou Silverlight. Actuellement, Native Client fonctionne sur une base x86 (uniquement). Mais il est disponible sur MacOS X, Windows, Linux. Par exemple le runtime Mac pèse tout de même 80 Mo... Le développement est actuellement prévu en Python. Il est possible d'utiliser ses outils classiques pour coder et compiler.

Site : http://nativeclient.googlecode.com

Composant
JViews 8.5 disponible

Sur le CD virtuel



Depuis fin novembre dernier, Ilog lance la nouvelle version de JView, la 8.5. Elle apporte de nombreuses nouveautés notamment sur Ajax. En particulier il ajoute un support des standards Web et des frameworks Ajax les plus populaires comme Apache Trinidad, JBoss Rich Faces, ICEFaces (de chez ICESoft), ou Facelets. Un meilleur support des serveurs de portlets a également été effectué pour fonctionner sous Pluto et LifeRay. Ceci est important dans la mesure où les développeurs d'applications Web choisissent leurs technologies serveur ainsi que des outils de productivité Ajax très tôt dans le cycle de développement. Au moment d'ajouter des fonctionnalités graphiques à leur application, il est crucial qu'ils aient un bon support du framework qu'ils ont choisi. Ces fonctionnalités Ajax sont disponibles pour tous les composants Web de la suite JViews (Diagrammer, Maps, Charts, Gantt et TGO). Cette version introduit aussi JViews Graph Layout for Eclipse, qui permet d'ajouter des fonctions d'agencement automatiquement de graphes et de diagrammes sous Eclipse. Ce produit permet d'enrichir des plug-in ou des applications RCP fonctionnant avec les couches EMF, GMF et GEF en y apportant une collection d'algorithmes de placement automatique. JViews Graph Layout for Eclipse se présente sous la forme d'un ensemble de plug-in à ajouter à l'environnement de développement d'Eclipse 3.3 ou 3.4.

agenda \\

JANVIER

• A partir du 12 janvier, **Ateliers techniques Blu Age,**
www.blueage.com/decouverte

• Les 19 et 20 janvier, Paris 17e , Palais des Congrès. **iLearning Forum :** le salon référence des offres de e learning en France. Le thème de cette édition : l'apprentissage intégré. http://www.ilearnforum.eu/

• Le 20 janvier, Paris 16 e Pavillon d'Armenonville,
Oracle Fusion Middleware Forum
Pour faire le point sur la stratégie Middleware d'Oracle.
http://www.oracle.com/events/fusion-middleware-forum/index.html

• **Séminaire Rogue Wave**
Du 20 au 21 janvier 2009 : Parsing de gros fichiers XML en C++ et Java.
Automatiser l'accès aux Bases de Données.
Exposer du code C++ existant en Web Services.
Consommer des Web Services dans du code C++ existant.
Rogue Wave Software - 40, rue des Vignobles 78400 Chatou
Tél 01 30 09 78 78

• Le 29 Janvier 2009, Paris 17e, Palais des congrès, Salon **emploi informatique les jeudis**
www.lesjeudis.com

• Le 29 Janvier 2009, Paris Automobile Club de France
Convergence et mobilité au service de la compétitivité de l'entreprise
http://www.it-news-info.com/invitation/Convergence-mobilite2009/index.html

FEVRIER

• Du 10 au 12 février 2009, Paris 17, Palais des congrès
Microsoft TechDays 2009
http://galilee.microsoft.fr/TechDays2009/Inscription-profil.aspx

Max 2008 – Milan

Sur le CD virtuel

Adobe se prépare à affronter Expression Blend !

Il y a plus d'un an, Adobe présentait déjà à Max (son événement annuel), Thermo, nom de code du futur Flash **Catalyst**. Dans cette vision, il s'agit de proposer une véritable tour de contrôle des projets entre développeurs et designers, bref d'avoir un outil d'intégration des éléments, encore au-dessus d'un outil comme Blend. Aujourd'hui, Thermo, alias Adobe Flash **Catalyst**, se présente différemment, un peu comme un outil à la Expression Blend. Il permet de créer rapidement des interfaces, des enchaînements d'écran à partir d'un projet illustrator, Photoshop ou d'un projet FXG. Disponible actuellement en pré-bêta, l'outil recèle encore pas mal de bugs et est disponible uniquement sous MacOS X. La bêta est attendue dans les prochaines semaines, la version finale, sans doute en octobre ou novembre prochain.

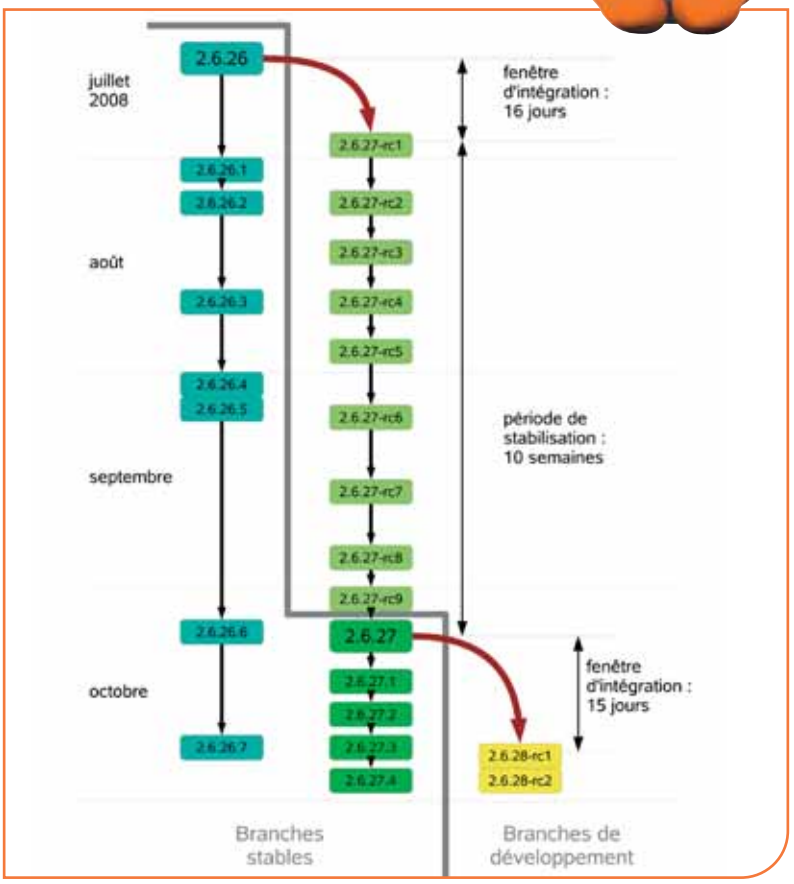
L'éditeur travaille aussi activement au futur de Flex avec la v4 et le futur Flex Builder. L'architecture de Flex 4 a été partiellement revue pour s'intégrer au futur Flash Catalyst. Sur **Flex 4**, notons aussi une refonte de l'architecture de composants et la volonté de pouvoir mieux découper les couches applicatives. Occasion aussi pour introduire de nouveaux formats comme le FXG, un format de fichier basé sur MXML et utilisable dans Flex, les outils CS4 et Thermo. On aura aussi droit au format FXP pour les projets Flex, partageable avec Thermo. A noter également que l'on pourra aussi créer des composants Flash. Flex Builder s'enrichit d'un nombre considérable de nouvelles fonctions pour le développeur sur le compilateur, l'éditeur de code, le refactoring, un mode debug bien plus puissant ou encore, la possibilité de monitorer intégralement son application…

Autre nouveauté présentée à Milan, le projet **Alchemy** (ex-FlaCC). Basé sur le compilateur LLVM et incluant une couche Posix, il permet d'utiliser du code C ou C++ dans la plate-forme Flash en " traduisant " le code existant en code ActionScript 3 utilisable dans Flash… L'éditeur prévoit plusieurs scénarios comme la logique métier, le transcodage audio vidéo ou encore les fonctions cryptographiques. Les performances sont un des atouts du projet.

Autre grosse nouveauté, la possibilité de coder des applications Flex directement sous Visual Studio ! Pour cela, deux possibilités : le plug-in **Tofino,** même s'il est très limité fonctionnellement ou **Amethyst.** Ce dernier est un IDE complet basé sur Visual Studio Shell. Techniquement et fonctionnellement, Amethyst est bien plus intéressant ! La version payante apporte un éditeur visuel très complet, la gestion du designer visuel par glisser-déposer, avec les réflexes des développeurs C# et VB. Il inclut l'Intellisense et le debug de l'IDE Microsoft.

Au cœur du noyau Linux

Le 10 octobre 2008 dernier, sur la liste de diffusion linux-kernel, Linus Torvalds annonce la disponibilité d'une nouvelle version du noyau Linux : 2.6.27 (1). C'est l'aboutissement de trois mois de travail. Un millier de développeurs de par le monde y ont contribué. Dans les semaines à venir, ce noyau deviendra la base des futures distributions Linux.



Alors que le noyau Linux vient de fêter ses 17 ans (2) et que la machine est désormais parfaitement huilée, le public ignore souvent les détails de l'organisation de cette fourmilière. Pour la majorité des utilisateurs de Linux, le noyau est une boîte noire entourée de mystère. Qui travaille sur le noyau Linux ? Comment cette communauté est-elle organisée ? Qui décide des évolutions ? Comment concilier un rythme de développement soutenu et les impératifs de stabilité inhérents au noyau d'un système d'exploitation utilisé par des millions de professionnels et de particuliers ?

Qui développe le noyau Linux ?

Le développement du noyau Linux est décentralisé et n'est pas lié à une entreprise. La Fondation Linux (3), sponsorisée par une cinquantaine de sociétés (4), synchronise les efforts. Elle emploie un certain nombre de développeurs (dont Linus Torvalds), de juristes et de personnels administratifs, mais l'immense majorité des développeurs sont répartis à travers le monde et employés par des centaines de sociétés différentes, ou contribuent au noyau Linux sur leur temps libre.

Bien que Linus Torvalds continue à s'occuper activement du noyau, il est devenu bien trop gros pour qu'il puisse s'occuper de tout. Le noyau a donc été découpé en une centaine de sous-systèmes, et chaque sous-système est sous la responsabilité d'une personne différente. Le découpage est effectué selon trois critères : le groupement fonctionnel (réseau, vidéo, son...), les standards matériels (PCI, USB...) et enfin les architectures matérielles (x86, IA-64, PowerPC...) Chaque responsable d'un sous-système est chargé d'en organiser l'évolution et d'en garantir la stabilité. Dans la pratique, il s'agit surtout de coordonner les efforts des développeurs intéressés par le domaine en question et de vérifier la qualité du code rajouté.

Les développeurs eux-mêmes n'ont pas de profil bien défini. La quantité de code contribué est extrêmement variable d'un développeur à l'autre. Le modèle de développement étant très ouvert – aucune inscription n'est nécessaire – il est fréquent de voir des contributeurs ponctuels disparaître aussi vite qu'ils étaient arrivés. En ce qui concerne les contributeurs réguliers, le rythme est très variable, de quelques modifications par an à plusieurs dizaines par semaine. La nature des contributions est également très variable, certains développeurs axant leur travail sur les performances, d'autres sur le support de nouveau matériel, d'autres enfin sur la stabilité. C'est la variété des intérêts et leur complémentarité qui fait toute la richesse de cette communauté.

Comment rentrer dans la communauté ?

En ce qui me concerne, je suis tombé dans le noyau Linux en 2002. À l'époque, je désirais simplement surveiller la température de mon système et la vitesse du ventilateur placé sur le processeur. Cette fonctionnalité n'était pas encore supportée pour mon système. Grâce à l'aide des

(1) http://lkml.org/lkml/2008/10/9/415
(2) http://www.linuxjournal.com/content/linux-turns-17
(3) http://www.linuxfoundation.org/
(4) http://www.linuxfoundation.org/en/Members

développeurs de cette partie du noyau, mon objectif fut relativement vite atteint. Il me sembla assez naturel de rester un peu pour aider les autres utilisateurs. De fil en aiguille, ceci m'amena à lire le code source des pilotes existants, à contribuer des pilotes pour le nouveau matériel, et finalement à devenir le mainteneur du sous-système en question en juin 2005. Je suppose que cette responsabilité n'a pas laissé les recruteurs de Suse Linux indifférents lorsque je leur ai proposé mes services. De fait, je suis depuis avril 2006 employé par Novell pour faire du support de niveau 3 sur les produits Suse Linux Enterprise, en particulier pour ce qui concerne le noyau.

Mon histoire est celle de beaucoup de contributeurs au noyau Linux. La majorité des développeurs principaux du noyau ont commencé cette activité sur leur temps libre pendant leurs études ou un peu après, motivés simplement par le challenge technique que représente le développement du noyau d'un système d'exploitation. Puis, la qualité technique de Linux et de son environnement et sa large disponibilité en ayant fait une base de choix pour de nouveaux systèmes d'exploitation, ces développeurs ont été recrutés par des sociétés informatiques pour lesquelles Linux représente l'avenir. Ceci leur permet d'être payés pour continuer à développer le noyau Linux, avec évidemment quelques contraintes supplémentaires selon les besoins de leur employeur. L'esprit de la communauté est resté celui des premiers jours : le mérite et l'excellence technique sont au pouvoir et l'on écrit encore du code pour le plaisir.

Qui prend les décisions ?

Le responsable d'un sous-système fait généralement autorité dans son domaine et dirige les débats qui s'y rapportent. De nombreuses listes de diffusion ont été mises en place pour faciliter les discussions entre les différents développeurs. Ces listes sont publiques et archivées, ce qui garantit le caractère ouvert et concerté des décisions qui sont prises. Lorsque des différences de points de vue existent, c'est toujours l'aspect technique des solutions qui est considéré en premier lieu. Si ce facteur n'est pas suffisant pour départager les propositions, le développeur le plus expérimenté ou le plus pugnace a généralement gain de cause.

Dans un certain nombre de cas, des décisions doivent être prises, qui dépassent les frontières d'un sous-système donné. Dans ce cas, la discussion est portée sur la liste de diffusion principale (LKML). Ces discussions peuvent être très engagées, comme le sont souvent les débats entre des personnes passionnées par leur sujet. Néanmoins le respect est de rigueur, et là encore le premier critère de décision est la comparaison technique des propositions.

L'expérience montre que les discussions sur des listes de diffusion peuvent parfois mal se terminer car l'interprétation de l'humeur des participants n'est pas forcément évidente et le ton peut donc monter assez vite sans qu'on arrive finalement à une solution. Ce phénomène est d'autant plus présent quand les intervenants ne se sont jamais rencontrés et ne connaissent donc pas leurs personnalités respectives. Or, la répartition géographique extrêmement étendue des développeurs du noyau rend ce cas de figure très courant. Pour pallier ce problème, la Fondation Linux organise chaque année une rencontre entre les développeurs les plus actifs du noyau Linux. Cette année, ce "Kernel Summit" s'est déroulé à Portland, aux USA (5). Pendant deux jours de discussions, les sujets d'actualité ont été débattus de vive voix dans une ambiance détendue. Les développeurs ont ainsi pu planifier bon nombre des modifications qui auront lieu au cours de l'année à venir. Se connaissant mieux, ils auront également plus de facilité à travailler ensemble à l'avenir.

Qui décide de la sortie d'une nouvelle version du noyau Linux ?

Une nouvelle version du noyau sort environ tous les 3 mois. Cette période est divisée en deux parties. Les deux premières semaines sont dédiées à l'intégration en masse de toutes les nouveautés qui seront intégrées dans le noyau à venir. Le reste – soit environ 11 semaines – est utilisé pour la stabilisation du code, c'est-à-dire la correction des défauts dans le nouveau code, ainsi que la recherche de régressions et leur correction. Pendant cette période, les modifications majeures sont théoriquement interdites, et leurs auteurs doivent patienter jusqu'à la prochaine fenêtre d'intégration.

A l'issue des deux premières semaines, une version candidate du noyau à venir est publiée avec le suffixe "-rc1" : première version candidate ("release candidate 1" en anglais.) Celle-ci n'a de candidate que le nom, et tous les développeurs savent par expérience qu'il faudra entre 7 et 9 versions candidates avant la version finale d'un nouveau noyau. Ces différentes versions sortent au rythme d'une tous les 9 jours environ. Chaque nouvelle version apporte son lot de corrections.

C'est finalement Linus Torvalds lui-même qui décide de la date exacte de sortie du nouveau noyau. Pour cela, il surveille le rythme de rapport de nouveaux problèmes (qui va en décroissant au fil des versions candidates.) Lorsque celui-ci est suffisamment bas, la décision est prise.

Comment sont gérés les problèmes rapportés et corrigés après la sortie d'un nouveau noyau ?

Il serait illusoire de penser qu'on puisse attendre d'avoir corrigé tous les problèmes avant de décider de sortir un nouveau noyau. Le noyau Linux compte pratiquement 9 millions de lignes de code et évolue très rapidement. La présence d'erreurs dans le code est inévitable. Le moyen d'améliorer la qualité n'est donc pas de retarder la sortie d'un nouveau noyau tant qu'il contient des erreurs – cela prendrait des années ou n'arriverait même peut-être jamais – mais de savoir corriger efficacement les problèmes qui sont découverts après la sortie d'un nouveau noyau, et de propager ces corrections aux utilisateurs le plus rapidement possible.

Pour ce faire, un processus connu

(5) https://www.linuxfoundation.org/events/kernel

sous le nom de "branche stable du noyau" a été mis en place en mars 2005. Son objectif est de collecter tous les correctifs importants pour une version donnée du noyau après sa sortie. Tous ces correctifs seront, bien évidemment, également présents dans la version suivante du noyau, mais il faudra jusqu'à 3 mois pour que celle-ci voie le jour, et l'on ne peut pas laisser les utilisateurs sans correctifs pendant aussi longtemps. Les branches stables comblent donc l'espace qui sépare les versions successives du noyau, afin qu'à tout moment il existe un noyau assez stable et assez sûr pour être utilisé par tout le monde. Les règles pour qu'un correctif soit admis dans une branche stable sont assez strictes, ce qui garantit l'absence de régression dans ces branches. Il convient d'observer que la date de sortie d'un nouveau noyau est donc quelque chose d'assez relatif et arbitraire. Des correctifs continueront d'y être apportés après coup, comme ils l'étaient juste avant.

Les branches stables du noyau sont utilisées comme base par toutes les distributions Linux majeures. Lorsqu'une mise à jour du noyau est décidée, les correctifs locaux à la distribution s'ajoutent aux correctifs communs apportés par les évolutions de la branche stable du noyau sur laquelle ladite distribution est basée. Cette approche permet de mutualiser les efforts de maintenance et d'anticiper les problèmes des utilisateurs.

Qui teste les nouveaux noyaux ?

Selon le modèle traditionnel du logiciel libre, les développeurs du produit sont aussi ses premiers utilisateurs. Aussi les premiers testeurs sont-ils les développeurs eux-mêmes. On peut même parler de tests en continu, car les développeurs sont tenus de suivre les derniers développements du noyau pour s'assurer que leurs propres innovations en cours de développement s'intègrent correctement.

Les tests effectués par les développeurs ne sauraient constituer à eux seuls une couverture suffisante. En effet, même si le nombre de développeurs est conséquent, il reste relativement faible en comparaison du nombre d'architectures, de périphériques et de systèmes de fichiers supportés par Linux. Une couverture plus large est donc assurée par les utilisateurs. Tous les utilisateurs ne sont pas prêts à prendre les mêmes risques ni à passer le même temps à rapporter des problèmes potentiels aux développeurs.

Les utilisateurs avancés de Linux qui désirent tester le futur noyau en avant-première peuvent ainsi choisir à partir de quelle version candidate ils le font. Les plus téméraires d'entre eux pourront attaquer dès la "rc2". Les plus raisonnables attendront peut-être la "rc6", qui est généralement une des dernières, voire la version finale. Les plus frileux attendront les premières mises à jour de la branche stable après la version finale. Ce modèle de test est un cercle vertueux : pour une version donnée du noyau, la qualité augmente au fil du temps, amenant d'autres utilisateurs à accepter de le tester. Ces nouveaux testeurs augmentent la couverture de test, améliorant encore la qualité, et ainsi de suite.

Statut des vieux pilotes

Avant même le début du Kernel Summit, Alan Cox qui ne pouvait s'y rendre cette année, lançait l'idée suivante : pourrions-nous, et devrions-nous, supprimer le support pour le matériel le plus ancien du noyau Linux 2.6 ? Le résultat deviendrait "Linux 3.0" (6) et pourrait être présenté publiquement comme une base plus propre pour les évolutions futures. Après discussion, peu de développeurs ont supporté cette idée, pour plusieurs raisons. La première est que la grande majorité des développeurs du noyau veulent que les évolutions de la version du noyau aient une raison technique. Or, la proposition d'Alan semblait tenir essentiellement du marketing. Dans la pratique, peu de pilotes peuvent réellement être abandonnés de la sorte, dans la mesure où l'une des forces de Linux est de fonctionner sur du matériel ancien sur lequel d'autres systèmes d'exploitation ne fonctionnent plus. Par ailleurs, on en vint à discuter l'intérêt même de la suppression de ces pilotes. Il s'avère que le coût de maintenance de ces vieux pilotes est pratiquement nul. Étant donné son âge, ce code est particulièrement stable, et le risque de mauvaise interaction avec du code plus récent est faible car il est rare d'utiliser du matériel vieux de 10 ans dans une machine moderne ou vice-versa. La décision prise a donc été, comme souvent lorsqu'on parle de Linux, celle du pragmatisme : tant que le coût de maintenance ne justifie pas la suppression d'un pilote, celui-ci est conservé. Ce n'est que lorsque maintenir un pilote coûte trop cher par rapport au petit nombre d'utilisateurs restant que celui-ci est supprimé. Il s'agit d'un critère technique comme les développeurs les aiment.

Numérotation des noyaux

Si cette question était rapidement tranchée, elle en réveillait pourtant une autre qui dormait depuis quelque temps déjà : le numéro de version des noyaux Linux a-t-il encore un sens, ou devrait-il être revu ? La façon de numéroter les versions date du temps où une période de plusieurs années séparait deux branches stables du noyau (par exemple la branche 2.2 et la branche 2.4.) La durée d'un cycle de développement pour un noyau actuel est de l'ordre de 3 mois, et de fait, il n'y a jamais eu de branche 2.8 ou 2.10. C'est maintenant le troisième nombre qui est incrémenté à chaque nouvelle branche. Cela signifie que tous les noyaux sortis depuis décembre 2003 ont un numéro de version en 2.6 et qu'il n'est pas prévu que cela change dans les années à venir. On a en revanche introduit un quatrième nombre pour désigner la branche stable qui suit la sortie d'un nouveau noyau. Les versions du noyau sont donc composées de 4 nombres dont 2 fixes (2.6) et deux seulement changent régulièrement. Cela semble inutilement compliqué. De nombreux développeurs aimeraient voir les deux premiers nombres fusionner. Le prochain noyau pourrait ainsi porter le numéro 2.7 et le suivant 2.8, ou encore 3.0 et le suivant 3.1. La

(6) https://lists.linux-foundation.org/pipermail/ksummit-2008-discuss/2008-August/000557.html

Perforce, le système de Gestion de Configuration Logicielle rapide

Le support technique Perforce

Des traitements rapides, des réponses précises

Nos équipes supports sont toujours disponibles afin de partager leurs expertises en personne - pas de réponses automatiques, de répondeurs ou de centres d'appel. Nos ingénieurs supports hautement qualifiés se font une fierté de traiter rapidement vos appels et de vous apporter des réponses précises.

Réaliser vos projets dans les délais nécessites des équipes supports prêtes à vous aider quand vous en avez besoin. Vous pouvez compter sur le système de GCL rapide Perforce et son support technique réputé, pour vous donner un atout gagnant.

PERFORCE
SOFTWARE

Téléchargez sans conditions une copie gratuite de Perforce sur www.perforce.com. Un service d'assistance technique gratuit est offert pendant toute la période d'évaluation.

branche stable issue de chaque noyau ajouterait un nombre, et l'on repasserait donc à des versions à 3 chiffres.
Cette vision des choses ne fait pourtant pas l'unanimité. Certains développeurs ne voient pas de problème avec la numérotation actuelle. D'autres, au contraire, pensent que quitte à changer quelque chose, une approche complètement nouvelle serait la bienvenue. En particulier, Greg Kroah-Hartman suggérait d'utiliser l'année comme premier numéro de la version, afin d'avoir une notion immédiate de l'âge d'un noyau en regardant simplement son numéro de version (7). Le deuxième nombre serait un compteur séquentiel remis à 0 au début de chaque année, ou bien le numéro du mois de sortie. Les détracteurs de cette approche notaient cependant qu'on ne connaît pas la date exacte de sortie du noyau avant sa sortie effective, et qu'il pourrait être nuisible pour la communication d'être incapables de parler du prochain noyau par son numéro.
À ce jour et après pas moins de 111 messages échangés sur la liste de diffusion, aucune décision n'a été prise. Il ne fait cependant guère de doute que ce sujet reviendra sur le devant de la scène dans les prochains mois, et ce jusqu'à ce que Linus Torvalds se prononce sur la question. Il est cependant très difficile de prévoir quelle sera sa réponse, tant il peut être conservateur sur certains sujets et ouvert à des changements majeurs sur d'autres.

Quand intégrer les nouveaux pilotes ?

Tandis qu'Alan Cox se focalise sur les vieux pilotes, Greg Kroah-Hartman, le responsable du projet "Linux Driver" (8), s'intéresse à une toute autre famille de pilotes : les pilotes expérimentaux pour le matériel le plus récent. Certains mainteneurs de sous-systèmes du noyau, dont je fais partie, sont très stricts sur les nouveaux pilotes qu'ils acceptent dans l'arbre principal du noyau. Leur argument principal est que la qualité du code est un atout majeur de Linux et qu'en acceptant des pilotes pas tout à fait prêts, la stabilité du noyau pourrait baisser et la réputation de Linux en pâtir. Par ailleurs, on a pu constater que certains contributeurs disparaissent rapidement dès que leur travail était accepté dans le noyau.
Cependant, force est de constater que cette approche peut décourager d'autres contributeurs, qui sont plus intéressés par l'écriture initiale du code que par sa mise en conformité avec les standards relativement élevés du noyau Linux. Dans ce cas, le code écrit risque fort d'être tout simplement perdu. Par ailleurs, retarder l'inclusion d'un pilote dans l'arbre principal d'un noyau rend plus difficile son utilisation par des testeurs potentiels. Or, les testeurs jouent un rôle important dans la qualité de Linux, et donc inclure les pilotes plus rapidement pourrait augmenter la qualité du code sur le long terme. C'est en tout cas le pari de Greg Kroah-Hartman, qui a demandé, et obtenu, la création d'un nouveau sous-système nommé "staging" dans l'arbre principal du noyau. Ce sous-système contient des pilotes qui nécessitent encore du travail avant de pouvoir être déplacés vers leurs sous-systèmes définitifs. L'avenir dira comment cette approche est accueillie par les développeurs et si elle portera ses fruits en terme de vitesse d'intégration du support pour le nouveau matériel et de qualité du code des pilotes.

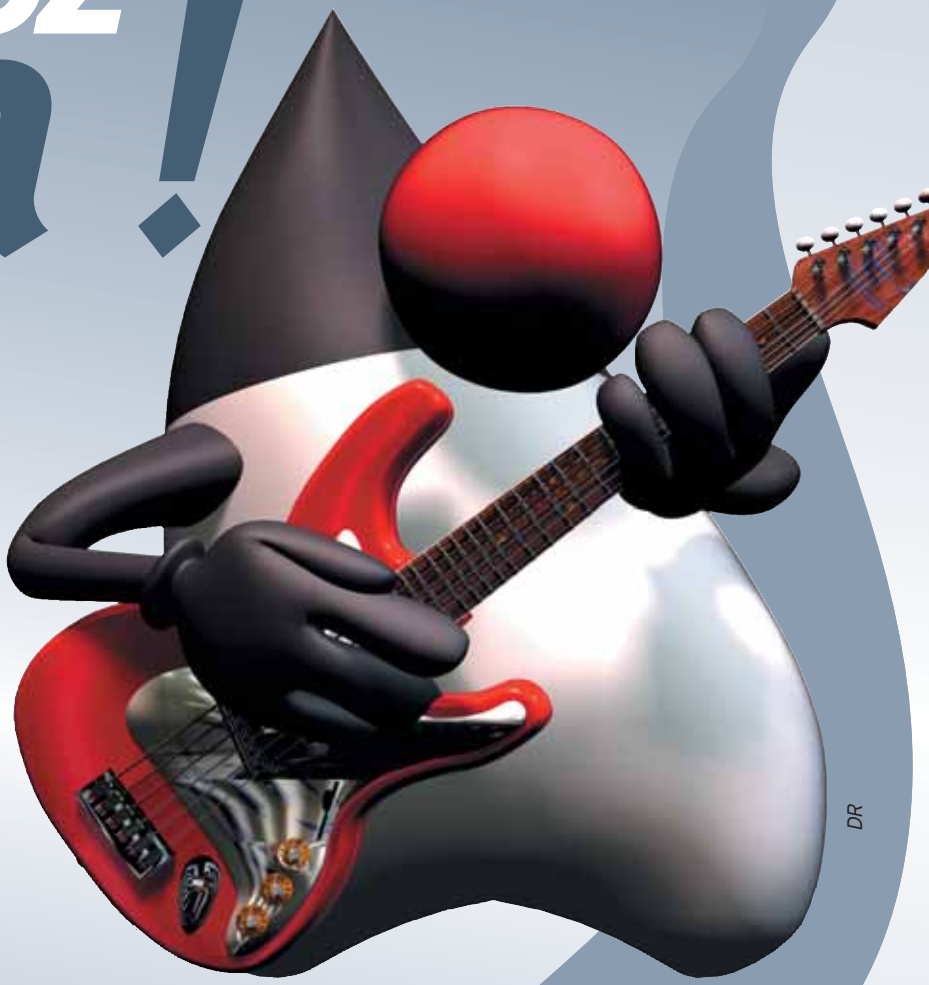
La durée du cycle de développement

Enfin, une des questions débattues lors du Kernel Summit cette année concernait la durée du cycle de développement des nouveaux noyaux. Initialement prévue de l'ordre de 2 mois par Linus Torvalds, elle est plutôt de 3 mois dans la pratique. Cela signifie qu'un développeur peut devoir patienter jusqu'à 5 mois et demi (2 mois et demi pour le noyau en cours de stabilisation et 3 pour le suivant) pour voir un changement majeur disponible pour le public, durée à laquelle s'ajoute le délai pour que les distributions intègrent le nouveau noyau à leur produit. Si cette durée est mise à profit pour tester le nouveau code, de nombreux développeurs s'accordent à considérer ce délai imposé comme trop important. Aussi a-t-il été proposé une réduction de la durée de cycle de développement à 6 semaines.
Le problème soulevé par Linus Torvalds est que la durée actuelle du cycle n'est pas le résultat d'un choix politique mais la simple conséquence de la nécessité de stabiliser le nouveau code. Cette durée est donc directement liée à la quantité de changements incorporés dans chaque noyau, celle-ci était elle-même liée à la fenêtre d'intégration de 2 semaines qui débute chaque cycle de développement. Cela signifie que le raccourcissement du cycle de développement devrait commencer par la réduction de la fenêtre d'intégration de deux à une semaine. C'est là que certaines dents ont commencé à grincer. En effet, la durée actuelle de la fenêtre d'intégration ne semble pas excessive. Il existe souvent des dépendances entre les modifications de divers sous-systèmes qui nécessitent que les développeurs coordonnent leurs efforts pendant la fenêtre d'intégration. À ceci s'ajoute la possibilité que certains mainteneurs soient en vacances ou occupés à d'autres tâches plus critiques pendant la fenêtre d'intégration, et cette possibilité augmenterait considérablement si la durée de la fenêtre d'intégration venait à diminuer.
Aucune décision immédiate n'a été prise à l'issue de la discussion. Linus n'a pas exclu l'éventualité d'une réduction de la fenêtre d'intégration, mais il a admis son scepticisme sur l'efficacité du procédé. Pour ma part, j'estime qu'une réduction du cycle de développement pourrait commencer par un plus grand respect de l'interdiction de modifications majeures après la fin de la fenêtre d'intégration. Pour l'heure, il est fréquent de voir des changements importants acceptés pendant une ou deux semaines après la fin de la fenêtre d'intégration, ce qui retarde d'autant le processus de stabilisation.

Jean **Delvare**
développeur noyau linux,
ingénieur logiciel chez Novell France

(7) http://marc.info/?l=linux-kernel&m=122411695231130&w=2
(8) http://www.linuxdriverproject.org/

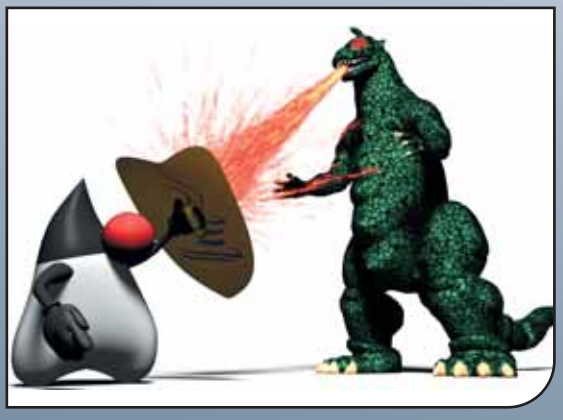
Boostez Java !



DR

Il est toujours intéressant et utile de faire un point régulier sur un langage et une plate-forme aussi utilisés que Java et Java EE. Avec la sortie de Java 6 Update 10 (Programmez ! n°114) et dans l'attente, de plus en plus longue, de Java 7 (pas avant fin 2009, voire en 2010), le monde Java bouge.

Dans ce dossier, nous verrons qu'il existe de multiples façons d'améliorer son quotidien avec Java. Nous allons ainsi aborder la configuration des EJB 3.x pour le fichier de configuration XML (simplifie grandement les choses), nous reviendrons aussi sur les techniques d'optimisation de son code Java, ce qui n'est jamais inutile. Nous aborderons les architectures Java EE et comment les comprendre, les réaliser. Nous parlerons du prochain Java, Java 7 et en particulier sur les accès (très intéressants), de programmation parallèle, un des défis que Java doit aborder rapidement et les premières briques techniques paraissent prometteuses. Avec les dernières évolutions de JDK, nous pouvons introduire dans les projets Java du code dynamique ! Les langages dynamiques sont pris en charge et ils le seront encore davantage avec la v7. Cette intégration est devenue une priorité aussi bien pour Java que pour .Net qui propose la même chose dans Silverlight 2.0 et dans le futur .net 4.0. Preuve à l'appui dans les pages suivantes !



DR

Nous aurons l'occasion durant l'année 2009 de revenir longuement sur de nombreux aspects du langage Java et ce, **dès le prochain numéro !**

François **Tonic**

Java 7 Fork/Join : Exploiter toute la puissance disponible

Les processeurs multi-core envahissent les ordinateurs ainsi que les serveurs et cette puissance accrue n'est encore que rarement bien exploitée. Java 7 intègrera parmi les nombreuses JSR, la JSR-166y, proposant ainsi un système de parallélisation des traitements permettant d'aller chercher cette puissance somnolente et de la mettre au profit de calculs gourmands.

Cela permettra de :
- gagner en temps de réponse (c'est assez évident à comprendre).
- réaliser des traitements plus ambitieux, sur des volumes de données plus importants (Cela est aussi rendu possible grâce au prix toujours décroissant de la mémoire physique, permettant le stockage de ces données volumineuses avec des contraintes d'I / O bien moins fortes que tout autre support de stockage).

Les problèmes du code

Jusqu'à récemment, le multithreading était un domaine dans lequel peu de monde osait se lancer, et pour cause : c'est un domaine assez complexe à maîtriser où les erreurs sont sanctionnées violemment. Le spectre du deadlock a hanté tous ceux qui s'y sont penchés. Et lorsqu'il était utilisé, le multithreading servait finalement assez peu à paralléliser réellement les tâches, mais plutôt à permettre, de façon assez simple, d'obtenir des traitements asynchrones. Ce qui était déjà un progrès :) (Je pense au SwingWorker...) Les choses se sont simplifiées avec l'arrivée de Java 5 et du package java.util.concurrent fournissant un ensemble de commodités pour s'abstraire un peu de la complexité initiale du multithreading. Les pools de thread, les queues, ... ont apporté un peu d'air frais. L'ExecutorService a simplifié l'exécution en parallèle de tâches, en s'affranchissant de la manipulation directe des Threads. Cette classe propose un pool de threads qui seront utilisés pour exécuter une liste de tâches lui étant soumises. L'objectif est atteint:cela fonctionne bien et de façon simple. Mais cette solution souffre de quelques limitations.

Tout va bien si les tâches sont indépendantes : une tâche s'exécute et une fois terminée, le thread l'ayant exécutée est libre pour traiter la suivante... mais si une tâche doit attendre le résultat d'une autre, qui s'exécute en parallèle, le thread qu'elle monopolise n'est pas rendu au pool tant que l'autre tâche n'est pas terminée et que la tâche courante n'a pas exploité le retour de l'autre. Apprécier la taille optimale du pool est toujours délicat. Le nombre de tâches parallélisables souhaité ne peut pas être trop important, car limité par le nombre de threads que cela induit. Trop de threads tue le thread ! La concurrence induite par un grand nombre de threads sur les ressources devient nuisible à la rapidité globale d'exécution.

L'apport de la JSR 166y

C'est là que la JSR 166y apporte un souffle nouveau... ou plutôt apportera, car le développement est toujours en cours. Le principe de base est la subdivision des traitements en plus petites unités de travail. Les traitements à un niveau de subdivision seront lancés en parallèle et les résultats obtenus en retour, agrégés. Ce principe est appelé fork / join. Chaque tâche, si sa taille est au-dessus d'un certain seuil, sera subdivisée en sous-tâches (fork) qui seront exécutées individuellement et parallèlement (ou subdivisées encore) puis agrègera (join) les résultats de chacune. Le comportement ici diffère de celui observable avec l'ExecutorService en cela que lors des périodes d'attente, le thread n'est plus monopolisé pour rien mais est affecté à l'exécution d'une autre unité de travail.

Prenons, pour illustrer ce fonctionnement, l'exemple proposé par Doug Lea : la résolution par récursivité (l'approche récursive n'est pas la plus performante, juste pratique pour la démo) de la suite de Fibonacci (dont la forme récurrente est : u(n+2) = u(n+1) + u(n), u(1) = u(2) = 1). Voici l'approche fork / join en Java :

```
class Fibonacci extends RecursiveAction {
[...]

    volatile int number;
    volatile int result;

[...]

    public void compute() {
        int n = number;
       if (n <= threshold) { // seuil de granularité sous
lequel on ne parallélise plus
            result = seqFib(n);
        } else {
           final Fibonacci f1 = new Fibonacci(n - 1); //
subdivision
            final Fibonacci f2 = new Fibonacci(n - 2);
            forkJoin(f1, f2); // invocation en parallèle
des sous-taches (fork & join)
           result = f1.result + f2.result; // agregation
des resultats
```

```
        }
    }

    public int seqFib(int n) {
        if (n <= 1) {
            return n;
        } else {
            return seqFib(n - 1) + seqFib(n - 2);


        }
    }

    public static void main(String[] args) {
         int groupSize = 2; // for example, try Runtime.
getRuntime().availableProcessors() too
        ForkJoinPool pool = new ForkJoinPool(groupSize);
        Fibonacci f = new Fibonacci(44); // for example
        pool.invoke(f);
        int result = f.getAnswer();
        System.out.println("Answer: " + result);
    }

[...]
}
```

L'exemple montre le fonctionnement de subdivision au delà d'un seuil ('threshold' dans le code), librement définissable, puis la parallélisation de l'exécution (via l'appel de la méthode *forkJoin(..)* de la classe RecursiveAction fournie par l'API fork / join) et enfin la consolidation des résultats. La méthode *main(..)* permet de voir le dimensionnement du pool d'exécution (groupSize, 2 threads ici, la methode *availableProcessors(..)* de la classe Runtime peut très bien fixer cette valeur) en charge de l'exécution des tâches parallélisables. Ce dimensionnement du pool ne doit pas être le simple fruit du hasard : il est optimal de définir comme nombre de threads le nombre de 'cores' disponibles (NOTE : ce nombre peut être inférieur au nombre physique) sur le serveur exécutant le code en question. Définir un nombre inférieur n'exploiterait pas toute la puissance disponible tandis qu'un nombre supérieur n'apporterait pas ou peu de gain : les threads en surnombre ne pouvant s'exécuter sans interruption d'un voisin actif (rappelons que le but du fork / join est de faire travailler tout le monde à 100% sans temps d'attente).

Chaque thread va être responsable de sa propre liste de tâches à traiter. Cependant, comme il peut avoir terminé son travail avant les autres threads du pool, il va pouvoir aller aider ses semblables. Voici comment un thread choisit une tâche. Une liste des tâches est implémentée dans chaque thread sous la forme d'une *double-ended queue* (appelée aussi 'deque'). La règle qui régit les accès des threads à cette structure est la suivante : Lors d'une opération de 'fork', les tâches créées par le thread sont empilées en tête de sa *deque*. Lorsqu'un thread est en attente d'une autre tâche pour une opération de type 'join', il va suspendre son traitement courant pour aller *dépiler* une nouvelle tâche depuis la tête de la *deque*. Si aucune autre tâche n'est présente dans sa propre *deque,* il ira 'voler' une tâche à un autre thread, mais cette fois par la queue de la *deque*. Ce mécanisme de consommation de tâches entre threads est appelé 'work stealing'. Le fait d'accéder par la queue à la *deque* de son voisin a plusieurs avantages : cela évite des problèmes de contention sur la tête : le voleur attaque la queue pendant que le volé s'occupe de la tête : les risques de contention sur une extrémité sont faibles. Comme voler une tâche est une opération qui ne survient qu'en cas d'épuisement de sa propre *deque,* la contention sur la queue de ces collections est aussi limitée naturellement car les occurences de vols sont moins fréquentes que les accès à la tête des *deque.* Autre avantage d'accéder à la queue de la *deque* du voisin : les tâches en queue de collections sont plus susceptibles d'être *subdivisables* car leur granularité est potentiellement plus importante que celles en tête (rappelez-vous : on empile les sous-tâches sur la tête). De ce fait, voler une grosse tâche donnera suffisamment de travail au voleur pour ne pas revenir de si tôt *piquer* chez le voisin.

Cette approche 'work stealing' est ainsi garante d'une répartition simple et efficace du travail entre threads, assez équitable en terme de charge et ne nécessitant que peu de synchronisation sur les ressources (du fait de la contention très limitée).

La parallélisation des traitements peut être utilisée dans ces scénarii bien plus fréquents que l'évaluation de la suite de Fibonacci. On peut citer parmi les tâches classiques pouvant tirer des bénéfices immédiats de cette approche fork / join : les tris, les recherches, les calculs comme des moyennes, détermination de maximum, minimum… sur des collections d'éléments.

Pour réaliser ces traitements sur des collections, l'API fournit une entité apportant un niveau d'abstraction des concepts de parallélisation encore plus important. Cette dernière va prendre en charge la subdivision, les déclenchements d'exécutions et les consolidations de résultats de façon transparente. La classe en question se nomme *ParallelArray* et va contenir un ForkJoinExecutor ainsi qu'un tableau de données sur lequel porteront les opérations.

Un apport important

Pour conclure, l'arrivée prochaine de Java 7 va ouvrir les portes, grâce à cette gestion de la parallélisation, à des améliorations sensibles des performances dans un certain nombre de programmes. Les gains obtenus peuvent être extrêmement élevés sur des serveurs dotés de beaucoup de cores (Des tests menés par Doug Lea montrent des gains allant de x15 à x28 sur un serveur doté de 30 processeurs). La bonne nouvelle aussi derrière cela est qu'il n'est pas exclu que ces modifications soient, au fur et à mesure, remontées à certaines parties existantes de Java (java.util.Arrays, …), offrant "gratuitement" aux utilisateurs des gains sensibles. Enfin, il existe des approches, plus complexes, de gestion de la parallélisation (algorithmes parallèles à grain adaptatif) qui permettent encore d'améliorer ce concept et qui finiront peut être implémentées dans des librairies tierces.

Quelques liens

http://jcp.org/en/jsr/detail?id=166

http://gee.cs.oswego.edu/dl/concurrent/dist/docs/index.html

David **MARTIN** - *architecte chez Sfeir*

Les outils de la Direction Informatique

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

L'INFORMATION SUR MESURE

Actu triée par secteur

Cas clients

Avis d'Experts

Etudes & Statistiques

Infos des SSII

Vidéos

Actus

Evénements

Newsletter

L'INFORMATION EN CONTINU

www.solutions-logiciels.com

Paramétrage des EJB3 en XML

Fini les fichiers de paramétrage XML! L'introduction des annotations, utilisées conjointement avec la pratique dite de *l'agressive defaulting*, ont permis de les éliminer. Les informations sont directement dans le code sous la forme d'annotations.

Cela ne doit pas faire oublier que les applications JEE sont prévues au départ pour permettre la répartition des responsabilités entre plusieurs rôles. Le rôle du développeur, qui réalise des composants, ou bien une application complète, et celui du déployeur, qui installe et paramètre l'application dans un environnement donné, voire souvent dans plusieurs environnements (tests fonctionnels, tests d'intégration, pré-production, production par exemple). Or le paramétrage suppose de pouvoir modifier les valeurs de certains paramètres de l'application sans avoir à recompiler. Heureusement, il a été prévu de satisfaire aux deux catégories de besoins : les valeurs spécifiées dans les annotations que le déployeur a besoin de modifier peuvent l'être en ajoutant un fichier de paramétrage XML. Ce fichier de paramétrage pourra ne contenir que les valeurs à surcharger, il sera inutile de redéfinir la totalité des paramètres.

Paramètres de type simple sur EJB3 session

Prenons une application JEE déployée sous la forme d'un serveur par région. Supposons que l'application dialogue avec une autre application, via un protocole quelconque, mais que ce serveur, bien entendu, est aussi régional. Par conséquent, le nom du serveur avec lequel l'application communique devra figurer dans un fichier de paramétrage texte, facile à modifier. Cela signifie-t-il qu'il faille revenir aux fichiers properties lus à la main depuis le code d'un EJB ? En dehors du fait que cette façon de faire n'est pas recommandée, elle engendre un supplément de code dans les EJB. En fait, les EJB3 ont conservé les possibilités de paramétrage disponibles dans les versions antérieures, la principale différence étant qu'en l'absence de ce paramétrage, la valeur indiquée dans le code est utilisée par défaut. La quasi-totalité des informations présentes dans les annotations peuvent être modifiées par simple paramétrage, sans toucher au code de l'application. Dans notre exemple, nous avons un EJB 3 session répondant à l'interface suivante :

```
import javax.ejb.Remote;
@Remote
public interface Service {
  void sendMessageToOtherApplication(String message);
}
```

associé à la classe :

```
import javax.annotation.Resource;
import javax.ejb.Stateless;

@Stateless
public class ServiceBean implements Service {

    String serverName = "testserver";
    public void sendMessageToOtherApplication(String message) {
        // ... pour illustration uniquement ...
  System.out.println("serverName = "+serverName);
    }
}
```

Dans cet exemple de code, le nom du serveur ne pourra pas être modifié au moment de l'installation sans recompilation. Ajoutons un simple tag devant l'attribut :

```
@Resource(name = "serverName")
    String serverName = "testserver";
```

Cela ne changera rien à la valeur de l'attribut, qui par défaut vaut toujours *testserver*. Ajoutons maintenant dans le répertoire META-INF de l'application un fichier ejb-jar.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
  version="3.0">
  <enterprise-beans>
    <session>
      <ejb-name>ServiceBean</ejb-name>
      <env-entry>
        <env-entry-name>serverName</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value>productionserver</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>
</ejb-jar>
```

La seule présence de ce fichier, lu et analysé par le serveur JEE, suffit pour demander à ce dernier d'injecter la valeur *productionserver* dans l'attribut *serverName* de l'EJB.

Autres paramètres

L'annotation *@Resource* est valable pour à peu près n'importe quelle ressource, à travers l'annuaire JNDI. Cependant, en dehors du cas des ressources de types simples, la correspondance n'est pas établie dans le fichier *ejb-jar.xml,* mais dans le descripteur de déploiement spécifique au serveur d'application.

En effet, le fichier *ejb-jar.xml* ne contient que des informations portables d'un serveur JEE à un autre, et si la ressource que l'on veut se faire injecter n'est pas définie dans le même fichier *ejb-jar.xml,* il est probable que le moyen de la référencer varie selon l'installation du container. Nous parlons ici de portabilité d'un serveur JEE à un autre, au sens implémentation JEE. Par exemple, Jonas et JBoss sont deux implémentations JEE, et une application JEE doit pouvoir

être déployée sur ces serveurs. Dans le cas de JBoss, les informations de mapping pour se faire injecter une ressource de type autre qu'un type simple sont dans un fichier *jboss.xml,* qui doit se trouver dans le même répertoire que le fichier *ejb-jar.xml.* Supposons que l'on ait maintenant besoin dans notre EJB de service de poster un message sur une file JMS. Nous avons besoin pour ce faire de la référence vers la file en question ainsi que d'une fabrique de connexions JMS.
Ajoutons sur notre EJB les attributs suivants :

```
@Resource(name="queue")
    Queue laQueue;

    @Resource(name="factory")
    QueueConnectionFactory factory;
```

Le fichier *jboss.xml* sera alors :

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.2//EN"
  "http://www.jboss.org/j2ee/dtd/jboss_4_2.dtd">

<jboss>
 <enterprise-beans>
   <session>
     <ejb-name>ServiceBean</ejb-name>
     <resource-ref>
       <res-ref-name>queue</res-ref-name>
       <jndi-name>queue/A</jndi-name>
     </resource-ref>
     <resource-ref>
       <res-ref-name>factory</res-ref-name>
       <jndi-name>QueueConnectionFactory</jndi-name>
     </resource-ref>
   </session>
 </enterprise-beans>
</jboss>
```

Ce fichier définit la correspondance entre le nom de la ressource tel que précisé dans l'annotation java et le nom JNDI de la ressource, les types doivent correspondre !

Configuration avancée des ressources

Les annotations *@Resource* peuvent être positionnées soit sur un attribut, soit sur un *getter* de l'attribut, soit directement sur la classe. Une même annotation ne peut pas figurer plusieurs fois sur un même élément. C'est pourquoi l'annotation *@Resources* a été prévue pour rassembler en fait plusieurs annotations *@Resource* que l'on souhaiterait toutes placer au niveau de la classe. L'annotation *@Resource* a un attribut *mappedName,* qu'il est recommandé de ne pas utiliser : son interprétation est susceptible d'être variable d'un container à un autre. Son usage fait perdre de la portabilité au code. Cet attribut se retrouve dans la définition de la ressource dans le fichier de configuration *ejb-jar.xml,* et pour la même raison de portabilité, il est préférable de déclarer la correspondance des ressources en utilisant le fichier de configuration spécifique au container JEE. Remarque : la majorité des serveurs JEE interprète le *mapped-name* comme étant le nom JNDI de la ressource.

Déployer plusieurs fois un même EJB Session

Pourquoi aurait-on besoin de déployer plusieurs fois un même EJB ? Dès lors que les paramètres peuvent être modifiés depuis le fichier de configuration, une application pourrait avoir besoin d'utiliser une version de l'EJB (session) avec certaines valeurs pour des paramètres et une autre version avec d'autres valeurs. Reprenons l'exemple d'un EJB session qui gère une communication avec une autre application (via JMS ou autre). Si en fait, en fonction de certaines règles de gestion, l'application destinataire est susceptible de varier, alors nous aurons besoin de disposer de plusieurs variantes de l'EJB session, une avec un paramétrage adapté pour chaque destination.
Le paramétrage se fait dans le fichier *ejb-jar.xml,* dans lequel il est possible de déclarer un EJB session plusieurs fois. Nous avons vu que la correspondance entre la classe de l'EJB et le paramétrage dans le fichier *ejb-jar.xml* est établie par le container à partir du nom de l'EJB (le nom de la classe par défaut). Pour déployer une seconde fois le même EJB dans la même application, le second déploiement doit avoir un nom différent. Ce nom ne peut donc plus être utilisé par le container pour faire la correspondance avec la classe, il faut donc ajouter dans le paramétrage le nom de la classe de l'EJB. Voici un exemple :

```
<session>
  <ejb-name>ServiceBean2</ejb-name>
  <ejb-class>test.ServiceBean</ejb-class>
  <env-entry>
    <env-entry-name>testParam</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>ServiceBean2 param</env-entry-value>
  </env-entry>
</session>
```

Avec ces lignes ajoutées dans le fichier *ejb-jar.xml,* la même classe d'EJB sera déployée une seconde fois, sous le nom ServiceBean2. Et l'EJB ServiceBean sera dans tous les cas déployé, même s'il n'est pas explicitement mentionné dans le fichier *ejb-jar.xml.* Il est donc possible de déployer la même classe d'EJB session plusieurs fois, ce qui permet d'avoir différentes configurations de paramétrage pour cet EJB. **Attention :** lorsqu'un EJB session est déployé plusieurs fois, il faut préciser lors de l'injection quel déploiement doit être injecté ! Dans le cas contraire, une erreur analogue à la suivante sera indiquée au moment du déploiement. Ainsi, pour se faire injecter une référence à un session bean dont la classe est déployée plusieurs fois sous des noms différents, il convient d'ajouter l'attribut *beanName* au moment de l'injection.

Conclusion

Le confort apporté au développeur par les annotations n'annihile pas pour autant la possibilité de paramétrer au moment du déploiement, et sans recompiler, les EJB. Les fichiers XML sont maintenant réduits au minimum en ne comportant que les écarts entre la configuration par défaut et la configuration installée. Une condition : que le déployeur ait à sa disposition une documentation suffisante sur les paramètres, leur signification et la manière de les configurer.

■ Dominique **Méra** - *Directeur de projets Objet Direct, filiale d'Homsys Group.*

Invocation de langages dynamiques en Java

Apparue avec le JDK 6, l'API Scripting permet l'intégration de langages de scripts au sein d'applications Java. Basée sur les spécifications de la JSR 223, elle se veut simple et légère pour le développeur, tout en étant extrêmement puissante, puisqu'elle offre aux concepteurs de langages tous les outils nécessaires pour l'invocation dynamique de ces derniers au sein de programmes Java. Dans la suite de cet article, nous nous intéresserons à la mise en œuvre de cette API à travers quelques exemples simples.

Le moteur de scripting Rhino de Mozilla, destiné à l'exécution de codes Javascript, est livré par défaut avec le JDK 6. Afin de nous concentrer uniquement sur l'utilisation de l'API, les exemples de scripts utilisés par la suite seront écrits en Javascript, mais il est bien évident que les parties Java sont quant à elles réutilisables avec d'autres implémentations de langages de scripts respectant la JSR 223.

Première invocation

La classe *ScriptEngineManager* est le point d'entrée de l'API côté développeur. Elle permet d'obtenir un certain nombre d'informations sur les moteurs de scripting installés, comme le nom à utiliser pour en instancier un ou la version supportée. Ainsi, il est possible de récupérer un moteur correspondant précisément à certaines spécifications. En outre, l'API expose des fabriques servant à récupérer une instanciation d'un objet ScriptEngine qui encapsule un moteur de scripting. Une fois ce dernier objet récupéré, il suffit d'utiliser sa méthode *eval* pour réaliser l'évaluation de codes du langage de script correspondant :

```
// Création du ScriptEngineManager
ScriptEngineManager manager = new ScriptEngineManager();
// Récupération d'une instance du moteur Rhino
ScriptEngine engine = manager.getEngineByName("rhino");
try {
     // Evaluation d'un Hello World ;)
    if(engine != null)
           engine.eval("print('Hello World !')");
} catch (ScriptException sexc) {
     sexc.printStackTrace();
}
```

L'exécution de ce code affiche un Hello World ! dans la console Java (figure 1). Ici, la méthode *eval* prend en paramètre un String mais il est également possible de lui passer en entrée un fichier contenant un script comme nous allons le constater.

Communication Java – Langage dynamique

Outre l'évaluation de langages dynamiques, l'API Scripting permet également une communication bidirectionnelle entre Java et le langage avec lequel on travaille. Cette communication peut être mise en œuvre de 2 manières différentes. La première consiste à utiliser les méthodes *put* et *get* de la classe *ScriptEngine* qui

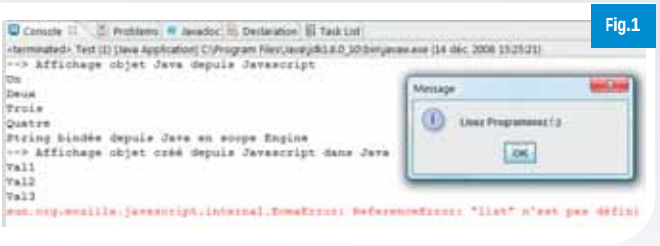


Fig.1

offrent respectivement la possibilité de passer au langage une variable et d'obtenir une variable définie dans le langage dynamique utilisé. Les variables passées pouvant être des classes Java de même que les variables récupérées peuvent être des objets Java créés depuis le langage dynamique. La seconde méthode est un peu plus évoluée puisqu'elle permet de spécifier la portée des variables à savoir si elles sont disponibles seulement pour un ScriptEngine donné ou bien si elles le sont pour l'ensemble des moteurs accessibles. Cette solution utilise l'interface *Bindings* qui permet de définir un ensemble de couples variable / valeur que l'on peut également passer en entrée d'une surcharge de la méthode *eval* du *ScriptEngine*. Ceci permettant de définir un contexte d'exécution spécifique qui remplacera le contexte du ScriptEngine courant pour une exécution donnée. Le code suivant met en avant ces différents points et le résultat de son exécution est présenté à la figure 1 :

```
List<String> list = Arrays.asList("Un","Deux","Trois","Quatre");
// Mise en place d'un objet Java
engine.put("list", list);
// Code permettant d'afficher le contenu de la liste en Javascript
String codeToEval = "var index; var val = list.toArray(); " +
     "println('--> Affichage objet Java depuis Javascript');" +
      "for(index in val){ println(val[index]); }";
// Evaluation
engine.eval(codeToEval);
// Récupération du contexte Engine du moteur
Bindings context = engine.getBindings(ScriptContext.ENGINE_SCOPE);
// Mise en place d'une variable
context.put("str", "String bindée depuis Java en scope Engine");
// Evaluation
engine.eval("println(str)");
// Création d'une liste Java depuis Javascript
engine.eval("importPackage(java.util); var l = Arrays.asList
```

L'agenda 2009 de cet **informaticien** est déjà bien rempli

Salon RECRUTEMENT Lesjeudis.com

INFORMATIQUE : DEVELOPPEMENT - AMOA - FINANCE

29 janvier - PARIS

Au Palais des Congrès - de 11h à 21h

- 12 février - LILLE
- 5 mars - NANTES
- 26 mars - GENEVE
- 2 avril - AIX-EN-PROVENCE
- 9 avril - PARIS
- 11 juin - TOULOUSE
- 17 septembre - BORDEAUX
- 24 septembre - PARIS
- Octobre - RENNES
- Novembre - LYON

Renseignements et pré-inscription sur **www.lesjeudis.com**

```
(['Val1','Val2','Val3']);");
// Récupération à l'aide de get
list = (List<String>)engine.get("l");
System.out.println("--> Affichage objet créé depuis Javascript
 dans Java");
for(String str : list){
     System.out.println(str);
}
// Création d'une JOptionPane depuis Javascript
engine.eval("importPackage(javax.swing);   JOptionPane.show
MessageDialog(null, 'Lisez Programmez ! ;)');");
// Evaluation du code de la liste avec un contexte spécifique
 => Erreur car liste inconnue
engine.eval(codeToEval, engine.createBindings());
```

Invocation de méthodes

Si elle ne permettait que l'exécution immédiate de code, l'API Scripting serait plutôt restreinte. Au lieu de cela, elle autorise l'évaluation de code contenant des méthodes, voire des objets du langage dynamique utilisé qui pourront ensuite être appelées depuis l'application Java quand cela s'avèrera nécessaire. Cette fonctionnalité se base sur l'interface *Invocable* et n'est pas forcément implémentée par tous les moteurs de scripting. De ce fait, il faut vérifier que l'instance de *ScriptEngine* que l'on récupère l'implémente bien avant de l'utiliser. Cette vérification faite, la classe *ScriptEngine*, castée sous forme d'objet *Invocable*, possède la méthode *invokeFunction* qui permet l'exécution d'une méthode du langage dynamique et la méthode *invokeMethod* qui permet d'appeler une méthode encapsulée au sein d'un objet. L'exemple suivant utilise un script contenu dans un fichier Javascript et invoque ensuite la méthode de l'objet défini au sein de ce dernier :

```
// Fichier test.js
importPackage(java.util);
var test = {
     list : Arrays.asList([11.8,12.85,13.06]),
     getList : function(){
          return test.list;
     }
};

// Code Java
// Evaluation du fichier test.js
engine.eval(new FileReader("test.js"));
// Vérification que le moteur implémente Invocable
if(engine instanceof Invocable){
     Invocable inv = (Invocable)engine;
     // Récupération de l'objet
     Object test = engine.get("test");
     // Invocation de la méthode
     Object val = inv.invokeMethod(test, "getList");
     if(val != null){
          for(Double d : (List<Double>)val){
               System.out.println(d);
          }
     }
}
```

L'exécution de ce code permet d'afficher le contenu de la liste définie dans le fichier Javascript test.js. D'autre part, lorsque l'on doit exécuter un script un certain nombre de fois il peut être intéressant de profiter de l'interface *Compilable* qui permet d'en obtenir une version compilée qui fera accélérer grandement son exécution. Comme pour l'interface *Invocable*, le moteur utilisé n'implémente pas forcément cette interface et il faut donc vérifier que c'est le cas avant de l'utiliser. Le cast du *ScriptEngine* vers le type *Compilable* permet d'accéder aux fonctionnalités de ce dernier et notamment la méthode *compile*. Cette dernière permet de récupérer un objet *CompiledScript* qui est la version compilée du script que l'on souhaite exécuter. Ensuite, l'appel à sa méthode eval va permettre son exécution.

Pour aller plus loin ...

De par sa taille réduite, cet article s'est concentré sur les fondements de l'API Scripting. Ainsi, les exemples proposés n'ont guère d'intérêt si ce n'est d'illustrer les basiques. Le lecteur intéressé par cette API, et désireux de voir des applications plus poussées, pourra trouver son bonheur sur la toile où les ressources commencent à devenir fournies. Il y trouvera également des explications permettant d'installer d'autres moteurs de scripting, chose qui se fait de manière relativement aisée. Le site http://scripting.dev.java.net/ présente un listing des différents moteurs actuellement disponibles. La liste des langages supportés est assez longue allant des classiques Ruby ou Python à des langages fonctionnels tels Haskell ou Scheme.

Figure 2 : Calculatrice post-fixée utilisant l'API Scripting

Conclusion

Les possibilités offertes par l'API Scripting sont nombreuses et ses champs d'application potentiels non moins importants. Elle pourrait ainsi être utilisée pour écrire des règles métier dans un langage plus simple à manipuler que Java, pour remplacer les fichiers properties pour la configuration d'une application et ainsi bénéficier de la puissance et de l'expressivité d'un langage de script ou encore pour prototyper rapidement une application Java. Cette liste est bien entendue non exhaustive et on pourrait encore imaginer réutiliser au sein d'applications Java des codes écrits dans d'autres langages plus appropriés pour certains types d'opérations. D'ailleurs, la JSR 292 qui a pour but de rajouter des fonctionnalités à l'API montre bien que Sun ne s'y est pas trompé et a bien compris l'importance qu'est amené à prendre cet outil.

Sylvain **Saurel** - *Ingénieur Etudes et Développement Java / JEE - Sopra Group agence Provence* - sylvain.saurel@gmail.com

Les bonnes pratiques d'optimisation du langage Java

Depuis l'apparition du langage Java il y a maintenant plus de 10 ans, les ingénieurs de chez Sun n'ont cessé de travailler à l'amélioration des performances de la JVM. Tant et si bien qu'à l'heure actuelle, Java en vient même à rivaliser sérieusement avec des langages plus proches du système que sont le C et le C++ ! Cependant, tout n'est pas parfait et pour de grosses applications, des problèmes de performance peuvent toujours se présenter. Pour faire face à ce dernier cas de figure, il peut être intéressant de s'attarder sur les bonnes pratiques d'optimisation du langage Java.

Quand doit-on optimiser de manière poussée son programme ? Voila la première question qu'il faut se poser avant d'aller plus loin dans la présentation de bonnes pratiques d'optimisation. En effet, comme l'a dit le célèbre Donald Knuth : *"Les optimisations prématurées sont à la racine de tout le mal"*. Par cette phrase, il voulait indiquer que la conception d'un programme ne doit pas être affectée par de petites optimisations qui ne feront au final gagner que quelques millièmes de secondes durant l'exécution d'un programme. Ainsi, un grand nombre de pratiques d'optimisation ne sont à prendre en compte que dans des cas extrêmes ou des goulots d'étranglements importants viennent à apparaître durant l'utilisation d'un programme.

Néanmoins, certaines d'entre elles ne nuisent pas au design des programmes et apportent un réel gain en termes de performances. De ce fait, il est intéressant pour tout bon développeur Java de les avoir en tête et de les mettre en pratique aussi souvent que possible. La suite de cet article va donc présenter brièvement quelques unes de ces bonnes pratiques d'optimisation regroupées en 5 thèmes principaux.

Strings

La classe String apportée par l'API standard de Java a pour caractéristique principale d'être immuable. Ainsi, une fois qu'un objet String a été créé, il ne peut plus être modifié. De ce fait, toutes les opérations proposées par cette classe vont nécessairement produire un nouvel objet String. Prenons l'exemple simple de la concaténation de différents objets Strings. Basiquement, on aurait tendance à utiliser l'opérateur + applicable à ce type pour la réaliser. Cependant, cette solution n'est pas vraiment la bonne puisque chaque concaténation de Strings présents dans l'affectation va donner lieu à la création d'un nouvel objet String ! Pour des concaténations très simples, cela ne pose pas de problème particulier de performance. Cependant, si le nombre d'objets Strings concaténés devient important et que l'opération se remette un grand nombre de fois, là alors il convient d'utiliser la classe StringBuilder de Java comme suit :

```
StringBuilder builder = new StringBuilder() ;
for(int i = 0 ; i < Integer.MAX_VALUE ; i++)
  builder.append(str[i]);
String result = builder.toString();
```

La méthode *append* de cette classe permet d'éviter les créations inutiles d'objets Strings et apporte un gain non négligeable à l'exécution. Afin d'accéder au contenu d'une String, l'API propose la méthode *charAt(i)* qui va renvoyer le char contenu dans l'indice i de l'instance sur laquelle la méthode est appelée. Cette solution d'accès peut être avantageusement remplacée par l'utilisation de la méthode *toCharArray* qui renvoie une copie du tableau de *char* représentant l'instance de String courante. Il est ensuite simple d'accéder à l'élément contenu à un indice donné via ce tableau.

Entrées / Sorties

Comme dans la plupart des langages, la gestion des entrées / sorties reste en Java aussi une cause majeure de dégradation de la vitesse d'exécution lorsqu'elle n'est pas bien adaptée. Le travail sur des données en mémoire dans la JVM est plus rapide que sur des données disque ou réseau. Ainsi, il est important de mettre à profit des techniques de buffering lors de la gestion des entrées / sorties aussi bien en écriture qu'en lecture. En Java, lorsque l'on travaille avec un InputStream, il faut donc préférer la méthode read(byte[]) à la méthode read() qui renvoie un simple byte. La taille du buffer contenant les bytes lus dépendant ensuite du contexte d'utilisation de la lecture, en gardant bien sûr à l'esprit qu'il faut toujours trouver un compromis entre occupation mémoire et temps d'exécution. Mieux encore, Java permet d'utiliser un buffering transparent en proposant des classes commençant par Buffered et en rajoutant ces fonctionnalités aux basiques classes de lecture et d'écriture. Pour un InputStream,

l'utilisation d'un BufferedInputStream bufferise ainsi l'appel à la fonction read() automatiquement.
Un dernier mot concernant les entrées / sorties sur l'API NIO de Java apparue depuis le JDK 1.4 et qui n'a cessé d'être améliorée depuis. Plus performante que sa devancière contenue dans le package java.io, elle rajoute en plus un certain nombre de fonctionnalités comme la copie de fichiers, le verrouillage de fichiers, ... Son utilisation ne doit pas forcément être automatique, mais lorsqu'une application nécessite une optimisation des opérations de lecture ou d'écriture, il ne faut pas hésiter à se tourner vers elle.

Collections

Les performances d'une structure de données et des algorithmes qui lui sont appliqués sont fortement liées à leur contexte d'utilisation. C'est pourquoi une bonne connaissance des différentes structures proposées par l'API Collections de Java ainsi que leurs contextes d'utilisation conseillés constituent un pré-requis inévitable pour obtenir des bonnes performances dans ce domaine. En outre, la classe *Collections* fournit un certain nombre de méthodes sur les collections concernant notamment le tri, la synchronisation ou la mutabilité. Les algorithmes qu'elles utilisent sont déjà optimisés et il va de soi qu'il est préférable de les utiliser dans son code plutôt que de réinventer la roue.
Les collections permettent d'ajouter des éléments sans avoir à se soucier de leur taille. Si le mécanisme d'extension dont elles bénéficient est transparent pour le développeur, il n'en reste pas moins coûteux à l'exécution. De ce fait, il est préférable de positionner à la création d'une collection si ce n'est sa taille définitive, au moins une taille qui permette de réduire le nombre d'extensions que la JVM aura à réaliser. Pour le parcours d'une collection, le choix doit se porter dans la majorité des cas sur un itérateur que l'on peut obtenir via la méthode *iterator* pour les List par exemple, ou de manière transparente en utilisant une boucle *ForEach.* En outre, le choix de la méthode d'itération dépendra des opérations que l'on souhaite faire sur les éléments de la collection durant son parcours.

Objets

Les objets sont le cœur même d'un langage orienté objet tel que Java. Ainsi, une utilisation adéquate de ces derniers doit permettre une amélioration des performances que ce soit en terme d'espace mémoire ou en terme de vitesse d'exécution. La JVM utilise un garbage collector évitant au développeur d'avoir à gérer la libération de la mémoire occupée par les objets qu'il crée. Ceci ne doit cependant pas enlever de son esprit le principe de fonctionnement du garbage collector. Ce dernier est lancé par la JVM lorsque cela est nécessaire et permet de libérer les objets n'ayant plus de références pointant sur eux. Afin de faciliter son travail, il est impératif de mettre à null les références sur un objet à chaque fois que l'on ne va plus l'utiliser. L'objet devient alors libérable et sera détruit au prochain passage du garbage collector.
La méthode *finalize()* héritée de la classe *Object* qui est censée être appelée par le garbage collector lors de la libération d'un objet n'est pas à utiliser pour plusieurs raisons. Tout d'abord, elle complique inutilement le travail du garbage collector puisqu'il est toujours possible de libérer les ressources utilisées par un objet ailleurs que dans cette méthode. Ensuite, elle n'est pas prédictible puisqu'on ne sait jamais quand elle sera appelée, ce qui complique fortement son utilisation. A éviter absolument donc !

Concurrence

La gestion de la concurrence et la parallélisation des tâches demeure une des grandes problématiques des applications d'informatique d'aujourd'hui et encore plus de demain. Les applications Java n'échappent pas à la règle ! Les différentes fonctionnalités liées à la concurrence amenées par l'API standard sont très puissantes mais peuvent se révéler pénalisantes si elles sont mal employées. Pour résoudre ce problème, le package java.util.concurrent est apparu depuis le JDK 5 et apporte des réponses concrètes et assez simples d'utilisation à la plupart des difficultés que le développeur peut rencontrer en matière de concurrence. En particulier, le framework Executor est à utiliser désormais pour manipuler les threads. En effet, il offre des fonctionnalités de gestion allant du simple thread aux groupes de threads mis en cache. L'exécution d'un thread via ce framework se passe comme suit :

```
ExecutorService executor = Executors.newSingleThread
Executor();
executor.execute(new Runnable(){….});
```

L'objet ExecutorService possédant des méthodes assez poussées permettant de bloquer un thread jusqu'à la fin de son exécution, ce qui était plus compliqué à réaliser auparavant. Ce framework propose également une implémentation semblable à la classe Timer mais qui s'avère plus puissante. Il offre enfin une alternative de plus haut niveau aux méthodes wait() et notify() qui bien que performantes font bien souvent chuter les performances par une mauvaise utilisation.

Sylvain **Saurel** - *Ingénieur Etudes et Développement Java / JEE - Sopra Group agence Provence* - *sylvain.saurel@gmail.com*

Java EE 5 : pour vos architectures

Java Enterprise Edition est apparue à la fin des années 90 et a apporté au langage Java une plate-forme logicielle robuste pour les applications d'entreprise. Remise en cause à chaque nouvelle version, mal comprise ou mal utilisée, concurrencée par les frameworks Open Source, elle a su tirer profit de ces critiques pour s'améliorer et trouver un équilibre dans sa version Java EE 5.

Pour certains, Java EE rime avec J2EE, et donc, avec lourdeur. Aujourd'hui, avec un retour à la modélisation objet et des serveurs d'applications légers tels que GlassFish, Java EE 5 a su trouver un nouveau souffle en faisant table rase du passé. Les annotations permettent de simplifier le développement et de limiter l'usage des fichiers de déploiement XML. Par exemple, pour rendre une classe persistante, il suffit de l'annoter avec @Entity et @Id.

```
@Entity
public class Adresse {
  @Id
  private Long identifiant;
  private String rue;

  // Constructeur, getter & setter
}
```

Vous retrouvez ce modèle de programmation objet dans la couche de service EJBs. Pour persister une adresse, il suffit de déclarer une classe @Stateless (c'est-à-dire un EJB sans état), de lui injecter un EntityManager (sorte de DAO générique), et d'utiliser la méthode persist (qui déclenchera un insert en base).

Java EE 5 pour tous les goûts

La force de Java EE 5 réside dans la richesse de ses APIs. Une application qui respecte les standards pourra s'exécuter sur différents serveurs d'applications. Les 22 spécifications constituant Java EE répondent aux besoins technologiques des architectures modernes. Les plus importantes sont les suivantes :

- **Servlets 2.5 :** Les Servlets permettent de recevoir des requêtes HTTP, de les traiter et de fournir une réponse dynamique au client
- **JSP 2.1 :** Java Server Page est une technologie qui permet aux développeurs de générer dynamiquement du code HTML (une JSP est compilée en Servlet)
- **JSTL 2.1 :** JSP Standard Tag Library est un ensemble de balises XML permettant d'ajouter de la logique dans les JSPs
- **JSF 1.2 :** Java Server Faces facilite la conception d'interfaces web, en gérant l'état HTTP ainsi que les événements entre client et serveur
- **EJBs 3.0 :** Les Enterprise Java Beans assurent les services de transactions et de sécurité. Il y en a de trois sortes : sans état (Stateless), avec état (Stateful) et recevant des messages (Message Driven Bean)
- **JTA 1.1 :** Java Transaction API permet de gérer les transactions

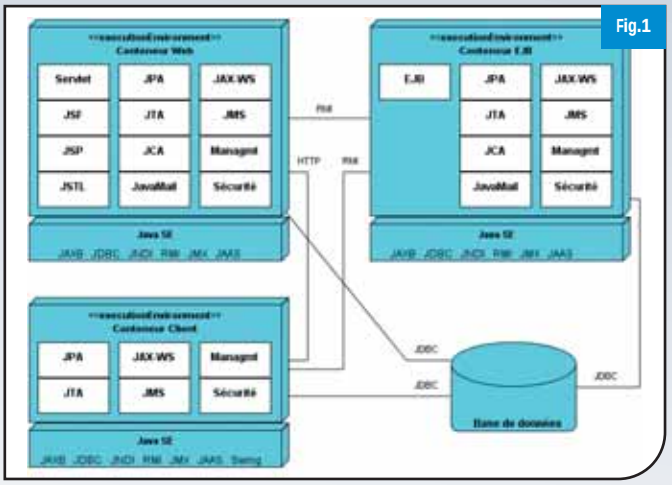


Fig.1

Conteneurs et spécifications Java EE 5

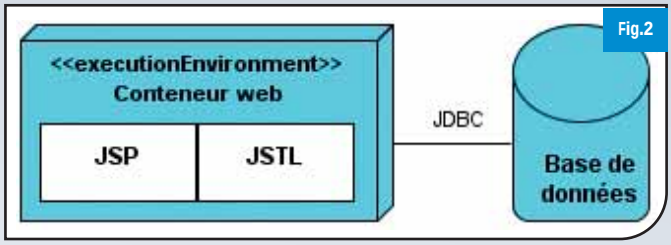
- **JPA 1.0 :** Java Persistent API permet d'assurer la persistance des objets
- **JMS 1.1 :** Java Messaging Service permet l'interopérabilité entre systèmes à l'aide de messages
- **JAX-WS 2.0 :** Java API for XML-based Web Services permet d'exposer et de consommer des services web

Certaines de ces spécifications nécessitent un conteneur pour s'exécuter. D'autres, comme JPA ou JMS, tournent dans une simple JVM. [Fig.1]

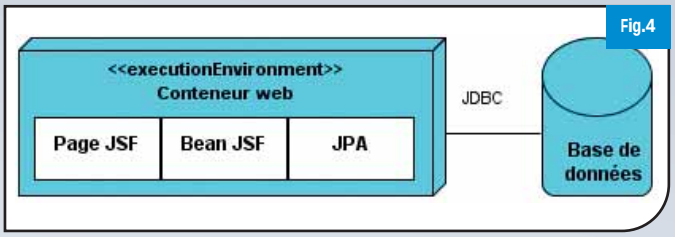
Le diagramme UML ci-dessus liste les conteneurs spécifiés dans Java EE 5, les interactions entre eux et la base de données, ainsi que les spécifications pouvant être exécutées par chacun.

- Le **conteneur web** (ou conteneur de Servlet) a la responsabilité d'intercepter les requêtes HTTP, d'invoquer une Servlet (ou JSP), de l'exécuter, de gérer son cycle de vie, et de renvoyer la réponse HTTP au client (ex. un navigateur).
- Le **conteneur d'EJBs** crée de nouvelles instances d'EJBs, gère leur cycle de vie et fournit des services tels que l'injection de dépendances, le transactionnel, la sécurité, la concurrence, la distribution…
- Le **conteneur client,** ou Application Client Container (ACC), apporte aux applications Java SE (ex. batch, Swing) des services de sécurité, de nommage, d'injection…

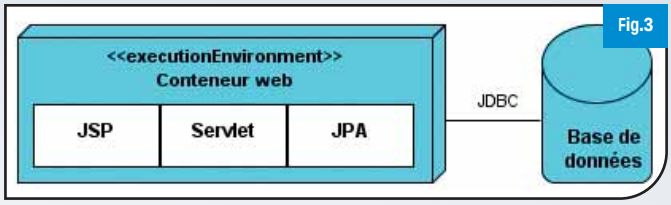
Tous ces conteneurs s'exécutant sur une JVM 1.5, les applications bénéficient donc aussi des spécifications JAXB (pour le marshalling XML), JDBC (pour accéder aux bases de données), JNDI (pour se connecter aux annuaires), RMI (pour les appels distants), JMX (pour la gestion de composants), JAAS (pour l'authentifica-



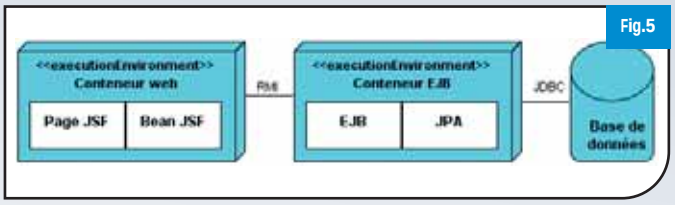
JSP accédant directement à la base de données



Pages JSF avec JPA



Architecture MVC avec JPA



JSF invoquant des EJBs

tion et l'autorisation) ou Swing pour les interfaces riches. Dans la pratique, les conteneurs Web, EJB et client sont implémentés par des serveurs d'applications tels que GlassFish, JBoss ou Weblogic. 22 spécifications, cela peut paraître beaucoup, compliqué et décourageant. Cependant, Java EE n'a pas inventé ses spécifications et les a développées au gré des besoins technologiques : pour répondre au transactionnel, à la haute disponibilité et à la distribution, il y a les EJBs, pour communiquer avec des services web il y a JAX-WS…

Vous avez dit architectures ?

La recherche de l'architecture unique est une quête du Graal. Il n'en existe pas une qui puisse répondre à tous les types d'applications, à toutes les contraintes techniques ou à tous les besoins de vos utilisateurs. Pourtant, il y a encore des cellules d'architectures qui passent leur temps à imaginer celle qui sera capable de répondre à tous les besoins d'aujourd'hui et de demain. Ceci, afin d'obtenir l'Architecture parfaite. Le résultat est souvent désastreux car, en plus d'être une quête onéreuse, les projets se voient alors contraints de suivre des règles gravées dans le marbre. Au lieu de se focaliser sur une Architecture avec un grand A, il faudrait plutôt s'intéresser aux architectureS avec un grand S. C'est là où Java EE 5 trouve sa force : en composant avec ces 22 spécifications.

Dans la Figure 1, on voit que certaines spécifications peuvent être utilisées dans des conteneurs différents (par exemple JPA est exécutable dans les conteneurs web, EJB et client). A l'opposé, certaines sont limitées à un seul conteneur (les Servlets, JSP, JSTL et JSF ne s'exécutent que dans un conteneur web). Selon les besoins de votre application, l'architecture peut aller de l'extrêmement simple au complexe.

Si votre application se résume à une simple interface web d'administration, mono-utilisateur, alors l'option JSP avec balises SQL de JSTL peut être une solution. Vos pages JSPs encapsulent alors des ordres SQL et accèdent directement à la base de données. Ce qui peut ressembler à une mauvaise pratique peut être tout à fait acceptable dans ce cas d'utilisation et vous permettra de développer votre application rapidement. Une autre solution est d'invoquer du code java directement dans la JSP et d'utiliser JPA pour la persistance. [Fig.2]

Si vous avez besoin de naviguer entre différentes pages et d'effectuer quelques traitements métier, vous pouvez mettre en place le design pattern MVC simplement à l'aide de Servlets. Celles-ci peuvent alors directement utiliser JBDC ou bien JPA pour accéder aux données. [Fig.3]

Pour une application avec une interface web plus riche, où l'on voudrait réutiliser des composants graphiques, des invocations Ajax…. il y a JSF. Cette spécification implémente aussi le pattern MVC. Les Beans de JSF peuvent alors accéder aux données directement en JDBC ou avec JPA.[Fig.4]

Certaines applications ont besoin d'un mode transactionnel. Dans ce cas, on peut directement utiliser JTA (Java Transaction API) dans les beans JSF et gérer soi-même les *commits* et *rollbacks*. Seul un conteneur web est alors nécessaire. Une autre solution est de développer une couche de service EJB et de laisser le conteneur gérer les transactions et la montée en charge. [Fig.5]

Pour interopérer avec d'autres systèmes, vous pouvez décliner exactement ces mêmes types d'architecture en rajoutant JMS pour envoyer des messages ou JAX-WS pour invoquer ou publier un service web. Ces deux APIs fonctionnent dans tous les conteneurs.

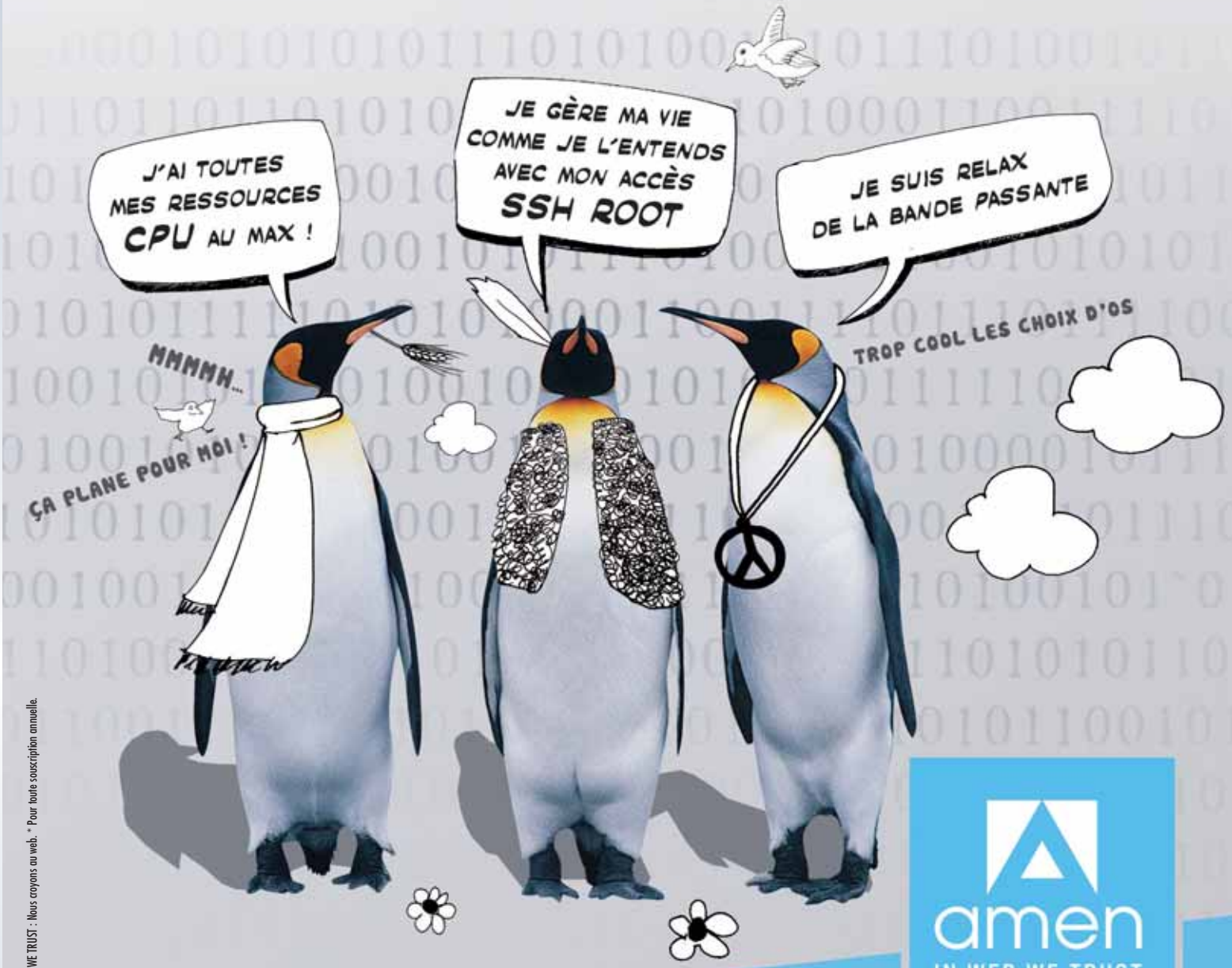
Des architectures agiles

“Les applications doivent être architecturée en couches”, tel est le vieil adage qui a du mal à être remis en question. On assiste alors à une frénésie de découpage (couche de présentation, de navigation, de logique métier, d'accès aux données…), de sur-utilisation de design-pattern et d'abstraction en tout genre. Les projets n'ont pas tous besoin d'avoir une architecture flexible et évolutive. Et de toute façon, il n'y a pas de code plus flexible que l'absence de code. Soyons donc minimalistes, respectons les besoins sans trop se projeter vers l'avenir. Une architecture doit démarrer modestement, avec juste l'infrastructure technique nécessaire pour soutenir l'activité de votre entreprise, et non résoudre les problèmes qui ne l'affectent pas encore.

■ Antonio.**Goncalves**

Consultant senior spécialisé dans les architectures Java EE. Son livre " Java EE 5 " *aux éditions Eyrolles détaille les spécifications ainsi que la manière de les assembler.*

NOUVEAU VDS+ d'AMEN :
le bonheur est dans le serveur !



amen
IN WEB WE TRUST

À PARTIR DE
5 € HT /MOIS
soit 5,98 € TTC/MOIS*

SERVEURS PRIVÉS AMEN : BÉNÉFICIEZ DE RESSOURCES GARANTIES QUI VOUS SONT PROPRES (PROCESSEUR, MÉMOIRE, DISQUE DUR...) TOUT EN PROFITANT D'UNE PLATEFORME INFOGÉRÉE 24H/24 - 7J/7.

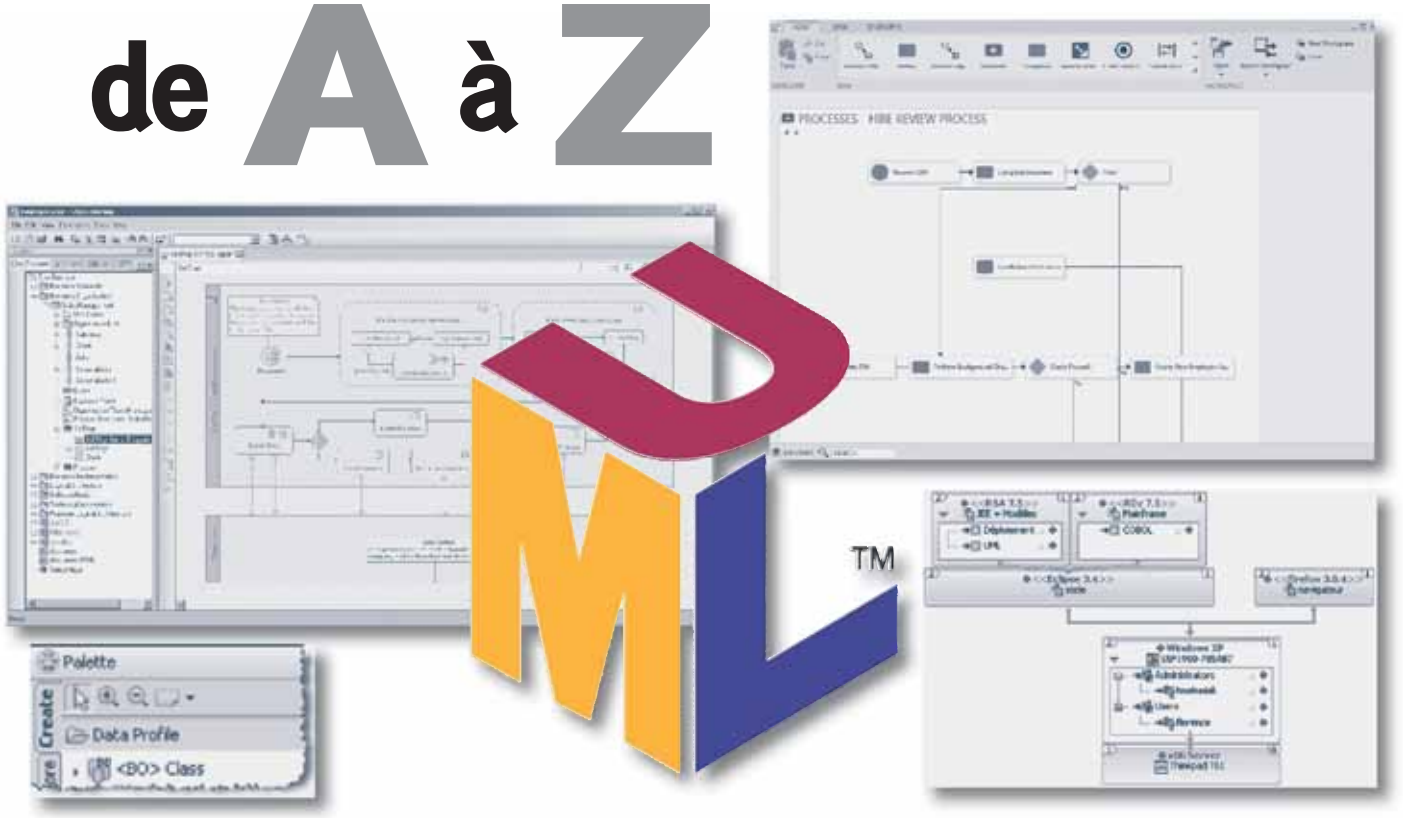
- Hébergement multi-sites/multi-domaines
- Interface d'administration : Plesk 8.6
- Systèmes d'exploitation : Fedora Core 8, Suse 10.3, Debian 4.0, Ubuntu 8.04 ou CentOS 5
- Part CPU minimum : de 1 à 6
- Mémoire garantie : de 256 Mo à 1 Go
- Espace disque : de 5 Go à 30 Go
- Bases de données : illimitées
- 1 adresse IP fixe
- Accès Root

PARTENAIRE INDUSTRIEL
entrepreneurs, faites le choix de l'économie numérique
www.econumerique.pme.gouv.fr

Pour plus de renseignements : 0892 55 66 77 (0.34 €/mn) ou www.amen.fr
NOMS DE DOMAINE - EMAIL - HÉBERGEMENT - CRÉATION DE SITE - E-COMMERCE - RÉFÉRENCEMENT

Sur le CD virtuel

La modélisation de A à Z



Le nouveau défi du développeur !

La modélisation devient incontournable pour le développeur, elle s'insinue en divers endroits de l'application comme la donnée, l'architecture cible, voire le déploiement. Le véritable défi est au niveau de la conception, du développement. Là, la modélisation fait des progrès rapides, notamment avec les outils de génération de code à partir de modèles, tout ce qui est développement orienté modèle. Gain de temps (supposé), moindre code, qualité en hausse (là aussi supposée). La démarche semble donc inévitable pour le développeur qui deviendrait un modeleur et non plus un développeur au sens actuel du terme. Mais est-ce la réalité ?

Dans les faits, la modélisation subit plusieurs obstacles. Le premier est la formation et la compétence dans la modélisation. Concevoir un modèle UML ou maîtriser un outil MD (Model Driven) demande une solide compétence et expérience. Ensuite, il faut savoir concevoir un modèle ni trop abstrait, ni trop complexe, ni incohérent. Dans les deux cas, on perd en qualité et l'étape suivante sera décevante. Mais la modélisation permet de se concentrer sur le code fonctionnel, le code métier, là où est véritablement la valeur ajoutée de votre travail. Car quel intérêt de coder une couche technique d'accès aux données, de mapping ou les interactions entre les écrans ? A terme, la modélisation doit permettre une meilleure productivité du développeur. Enfin, on l'espère tous !

Malgré tout, la modélisation reste souvent un monde hostile pour le développeur. Et il y a de quoi perdre son diagramme de classe car le marché est particulièrement diversifié. Vous y trouverez les modeleurs UML " pure player ", des modeleurs intégrés à des IDE, des outils de génération de code, des outils orientés MD, etc. Prenez simplement le Model Driven, la dernière journée MD Day fut un véritable succès prouvant l'intérêt pour cette technique mais on se rend compte de la complexité de l'offre et surtout de la diversité de l'approche MD avec le MDA, MDD, MDT, etc. Et les utilisateurs sont tout de même hésitants après les promesses non tenues des outils MDA de première génération. Et en France, Merise a la vie dure...

Dans ce présent dossier, nous allons démystifier la modélisation aussi bien MD que UML, avec l'aide de quelques-uns des meilleurs experts français dans le domaine ! Le dossier oscille entre la théorie (en modélisation il est vital de comprendre la théorie) et la pratique. Nous vous dirons pourquoi et comment utiliser UML, comment choisir son Model Driven, les bonnes pratiques UML, bien débuter en UML. Nous verrons aussi quel modèle pour quel développement. Nous ferons un point sur le marché, les tendances.

■ François **Tonic**

Le challenge de la modélisation pour le développeur

Ed Merks, l'un des grands gourous de la modélisation au sein d'Eclipse, énonçait un concept simple durant sa session au MDDay Paris : *la modélisation ne remplacera pas la programmation.* Et il est vrai que si aujourd'hui, la tendance est de vouloir générer le plus de code possible, voire même toute son application, maîtriser le code et la programmation demeure la base. Et Ed d'ajouter que *la modélisation est restrictive et peut limiter la créativité du développeur.*

Le modèle apporte en effet une rigueur dans la conception, avec du formalisme, la nécessité de définir au mieux les spécifications de son application, etc.

La modélisation peut être vue comme un élément complémentaire pour le développeur, mais encore faut-il maîtriser le modèle... Et il ne faut pas omettre un élément : la maîtrise de l'UML, d'un outil Model Driven, nécessite de la formation, de la pratique. Il faut savoir trouver un compromis dans la granularité du modèle. Faut-il confier toute son application au modèle ? Tout va dépendre du type d'application. Dans un projet d'intégration de différentes applications, de différentes données, ou dans des applications métiers, la modélisation peut effectivement aller très loin, avec un minimum de retouches. Aujourd'hui, il faut considérer le modèle comme le moyen pour le développeur de ne plus s'occuper du code technique, qui apporte finalement un code de commodité pour les accès aux données, les enchaînements entre les écrans, etc. La réelle valeur n'est pas là. Mais encore faut-il que le code généré soit de qualité, capable de répondre à des métriques et des pratiques précises. En France, un autre constat s'impose *" la génération a mauvaise presse ",* précise Nicolas Romanetti (Jaxio).

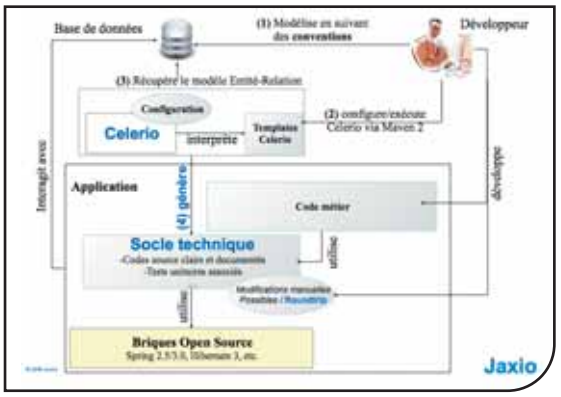
La modélisation influence le métier de développeur

Cette affirmation est une réalité. Car la génération de code oblige aussi à repenser, au moins partiellement, le rôle et les prérogatives des développeurs. Il faut alors repenser leur travail au quotidien. Le métier du développeur évolue même si on refuse de le voir. *" Le développeur va maîtriser (ses outils). Il est déjà intégrateur avec l'utilisation de briques open source ",* poursuit **N. Romanetti.**

Les DSL : l'autre modèle ?

Dans ce présent dossier, nous avons volontairement pris le parti de l'UML et des méthodes MD (Model Driven). Le Domain Specific Language est cependant une réalité, même si aujourd'hui il a du mal à se faire une place, faute d'un manque d'outils ou d'outils peu ergonomiques. L'avantage du DSL est de pouvoir créer un langage spécifique à des besoins précis, on est encore au-dessus de la " simple " modélisation. L'autre avantage du DSL est que l'on peut en définir autant qu'on le souhaite, les personnaliser. Et en théorie, le DSL est moins complexe que l'UML pur. Mais rien n'empêche de mélanger la double approche : DSL + UML. Ensuite, on retrouve l'opposition traditionnelle entre les approches bottom-up et top-down. Mais là non plus il ne faut pas avoir d'a priori, l'approche doit correspondre à votre projet. Il existe plusieurs avantages dans le DSL : on est constamment guidé, il est validé en continu, l'intégration est excellente avec les IDE, etc. Mais le DSL sera essentiellement lié à sa plate-forme, donc d'une portabilité douteuse.

On peut se demander s'il faut une génération de code à partir d'un modèle ou faire de l'exécution de modèle. Et également s'il faut un modèle " générique " pour l'interface, le code métier, la persistance ou un modèle par domaine. Il n'y a pas de réponse unique. Là encore cela va dépendre des projets. Au-delà de la question : faut-il prendre de l'UML, du MD, du DSL, il faut aussi se poser celle de l'outil. Et c'est un choix crucial, même si dans le monde UML, XMI doit permettre d'échanger des modèles entre modeleurs (attention tout de même à la bonne interopérabilité). Il ne sera pas prudent de changer d'environnement en cours de route.



Industrialisation

Comme vous le verrez dans les différents articles de ce dossier, la modélisation permet au moins une chose : l'industrialisation du développement. Nous rentrons ainsi dans les usines à logiciels ! Et ce n'est pas un mal. Mais n'oubliez pas une chose : la qualité de cette industrialisation dépend de votre maîtrise des techniques, des outils, des spécifications du projet. C'est aussi un changement dans le travail au quotidien où la notion de réutilisation n'est pas un mot obscur mais une nécessité. Même si le retour sur investissement (notre vénéré ROI) peut être difficilement mesurable...

■ François **Tonic**

A suivre...

Nous poursuivrons notre exploration de la modélisation, des modèles dans le prochain numéro (n°116, sortant fin janvier 2009). Nous y aborderons notamment les DSL et OSLO.

Mes premiers pas en modélisation UML !

Les trois lettres UML sont bien connues des développeurs. Elles signifient : Unified Modeling Language. Il s'agit d'un langage de modélisation graphique et textuel permettant, par exemple, de spécifier et de concevoir un système. Dans sa version actuelle, UML propose 13 vues graphiques d'un même système : 13 diagrammes. Face à cette " complexité " les développeurs cherchent, bien souvent, des exemples concrets pour démarrer en douceur.

Dans cet article, découpé en 5 parties, nous allons aborder de manière pragmatique l'utilisation d'UML pour modéliser, représenter et générer le code d'une application fictive.

Description du contexte applicatif

Le cas d'étude de cet article est une application de robotique. Le robot à piloter est simulé dans l'environnement 3D Gazebo (1). Cet environnement est libre. Il permet de simuler des pièces mécaniques et des capteurs associés à cette mécanique. Des modèles de robots ainsi que des APIs logicielles, prêtes à l'emploi, sont ainsi fournis par le simulateur pour rapidement mettre en place une application de robotique. Ces APIs sont fournies par un serveur nommé *Player.* Pour piloter un robot, l'application s'interface par l'intermédiaire de sockets TCP avec *Player*.

D'un point de vue mécanique, le robot utilisé dans cet article est un robot à quatre roues (2). Il est composé de huit capteurs de distance (des télémètres) et de deux actionneurs (un pour la roue avant droite et un pour la roue avant gauche).

D'un point de vue logiciel, la connexion avec le serveur *Player* est réalisée par la librairie *Client.* Les télémètres sont accessibles par la librairie *Sonar.* Enfin, les actionneurs sont accessibles par la librairie *Position2d.* Cette dernière permet, d'une part, d'assigner une position cible et, d'autre part, de connaître la position actuelle du robot. Cette position est alors décrite dans un repère à deux dimensions. La figure 1 résume ce contexte de développement.

Dans cet article, le système développé est un contrôleur, c'est-à-dire une entité logicielle capable de piloter le robot. Ce contrôleur doit, d'une part, s'interfacer avec les entités simulées (les librairies *Client, Sonar* et *Position2d)* et, d'autre part, permettre l'assignation d'une position cible au robot. Cette assignation est alors ordonnée par un utilisateur physique ou logiciel. A partir de ce cahier des charges, UML est utilisé pour spécifier, concevoir et implanter le contrôleur.

Spécification de l'application

Parmi tous les diagrammes UML, celui le plus approprié pour l'étape de spécification est le diagramme des cas d'utilisation. Du point de vue de l'architecte logiciel, chaque cas d'utilisation représente un service rendu par le système. L'environnement dans lequel évolue ce système est alors modélisé sous la forme d'acteurs. Un acteur peut être une personne physique, l'utilisateur, ou une entité logicielle extérieure au système développé, une librairie logicielle par exemple.

Dans la figure 2, le cahier des charges du contrôleur a été traduit en cas d'utilisation. Le contrôleur est spécifié comme un système UML puisque c'est lui qu'il faut développer. Les entités externes à ce contrôleur sont décrites comme des acteurs. L'utilisateur et les librairies offertes par le simulateur sont ainsi des acteurs. De manière pragmatique, les acteurs à gauche du système utilisent le contrôleur. Les acteurs de droite sont utilisés par le contrôleur. Ainsi, l'acteur utilisateur utilise le contrôleur. L'acteur client gère la connexion avec le simulateur et met à

Figure 1 : Contexte de développement

Figure 2 : Fragment d'un diagramme de cas d'utilisation du contrôleur

Fig.1

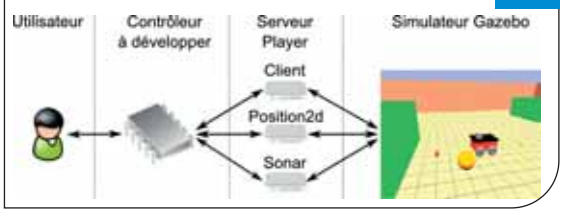
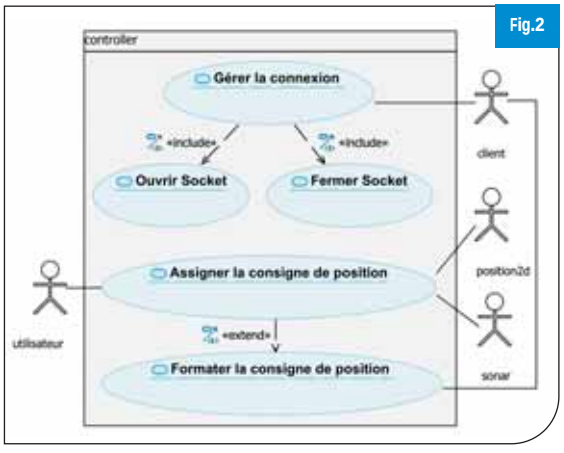


Fig.2



1 - http://playerstage.sourceforge.net/
2 - *Robot de type Pionneer 3dx :* http://www.activrobots.com/ROBOTS/p3dx.html

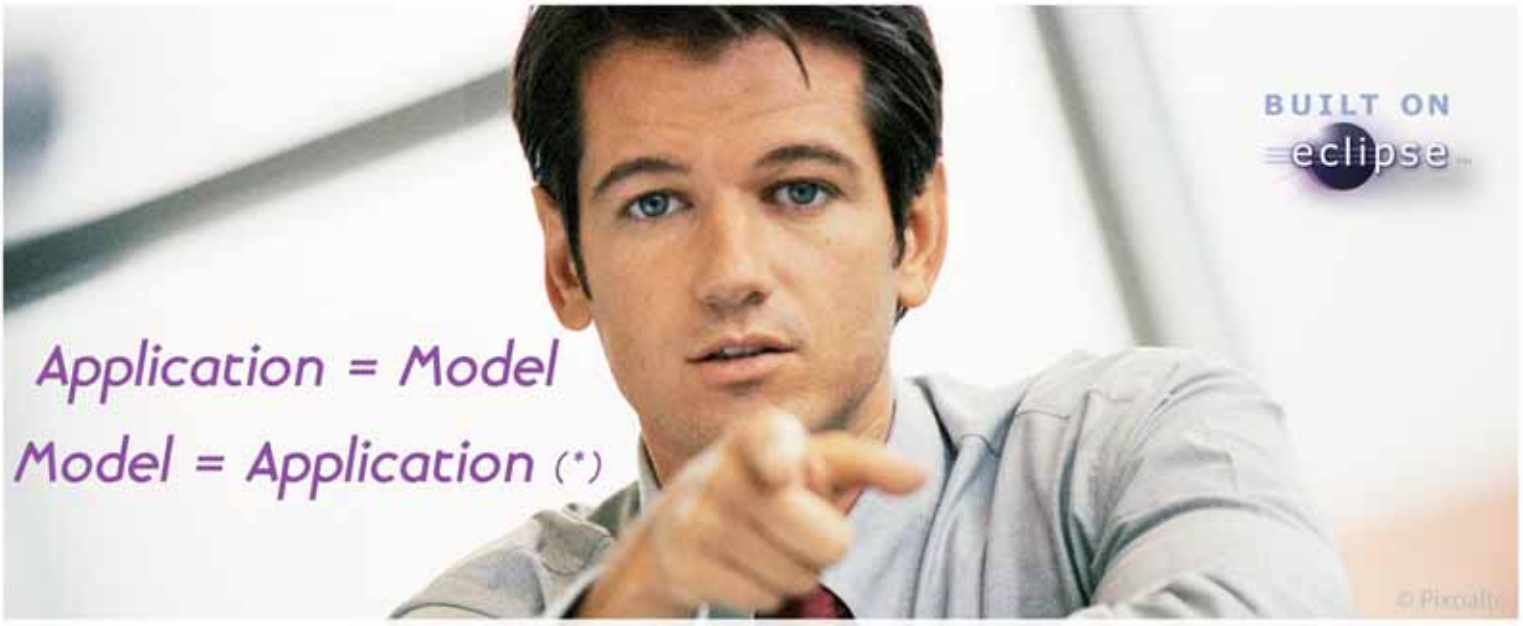
BLU AGE

AGILE MODEL TRANSFORMATION

Sublimez vos modèles avec BLU AGE® Edition 2009 !

Avec BLU AGE®, valorisez enfin vos connaissances en UML !

Rejoignez la révolution des développements agiles !



Venez découvrir BLU AGE® Edition 2009

8 ATELIERS TECHNIQUES TOTALEMENT GRATUITS !

A partir du 12 Janvier 2009

A Nantes, Bordeaux, Toulouse,
Lille, Paris, Lyon, Strabourg, Marseille.

Un après-midi pour :

- Découvrir les fondamentaux du Model Driven Developement™,
- Vous initier aux techniques de modélisation UML et de génération automatique de code et d'applications,
- Passer de la théorie à la pratique en générant votre première application "école" avec notre atelier BLU AGE®.

Informations et Inscription :

01 56 05 60 91

www.bluage.com/decouverte

Construit sur Eclipse, BLU AGE® Edition 2009 transforme instantanément vos modèles UML en applications JAVA EE et .Net.
Implémentation pragmatique du MDA, BLU AGE® outille vos développements agiles.

BLU AGE est cofinancé par l'Union Européenne.
L'Europe s'engage en Aquitaine avec le Fonds européen de développement régional.

(*) L'application est le modèle, le modèle est l'application

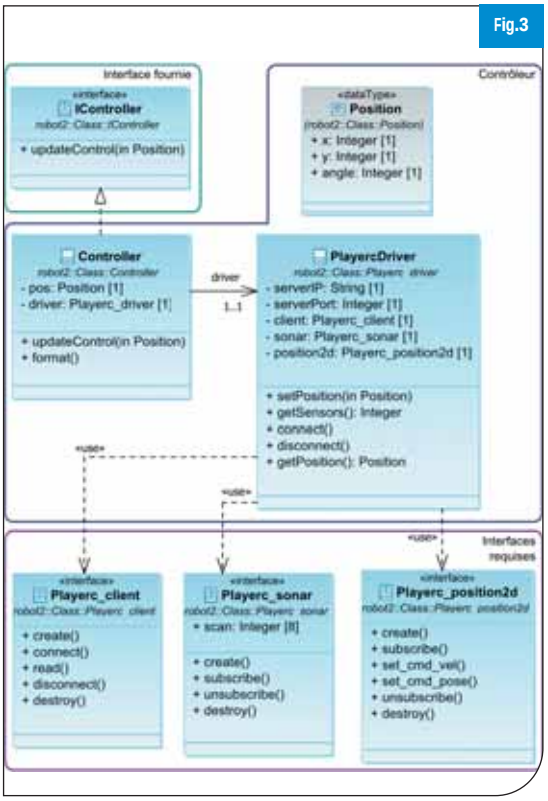
jour les données des capteurs. L'acteur *position2d* permet d'assigner une position au robot. Enfin, l'acteur *sonar* rend compte au contrôleur des huit mesures télémétriques. Le contrôleur doit ainsi gérer les connexions avec le client. Il doit aussi assigner, à l'acteur *position2d,* la nouvelle position cible fournie par l'utilisateur. Cette consigne est une position relative par rapport aux coordonnées initiales du robot. Par extension, le contrôleur devra donc préalablement calculer la consigne réelle à affecter. Cette extension est modélisée par une relation d'extension entre les cas d'utilisation Assigner la consigne de position et Formater la consigne de position. Notez que les verbes employés dans les cas *d'utilisation* sont tous des verbes à l'infinitif. Ils expriment tous des fonctionnalités attendues.

A partir de ces spécifications, il est désormais possible de concevoir les fonctionnalités attendues de cette application.

Conception de l'application

La conception d'une application passe par la conception structurelle et comportementale de l'application. Dans ce cas d'étude, le diagramme de classe est utilisé pour décrire la structure du contrôleur. Le diagramme d'activité est utilisé pour décrire le corps des opérations de ce diagramme de classe. La figure 3 illustre un fragment du diagramme de classe.

Figure 3: Fragment d'un diagramme de classe du contrôleur



Afin de simplifier la conception du contrôleur lui-même, celui-ci est scindé en deux classes : *Controller* et *PlayercDriver.* La première assigne et formate la position cible voulue. Cette position est modélisée sous la forme d'un type de donnée (DataType) constitué d'une abscisse (x), d'une ordonnée (y) et de l'angle dans lequel est positionné le robot à cette position (angle). Cette classe *Controller* est associée à un *PlayercDriver,* c'est-à-dire un pilote dédié au serveur *Player.* Ce pilote gère les connexions avec les librairies du serveur (voir cas d'utilisation Gérer la connexion). Il possède pour cela des attributs *serverIP* et *serverPort* pour mémoriser l'adresse IP et le port du serveur.

A partir du diagramme de la figure 2, tous les acteurs ont été transformés en des interfaces. L'acteur utilisateur est une interface, *IController,* fournie par le contrôleur. Elle est le point d'entrée de l'utilisateur pour ordonner la mise à jour de la consigne (updateControl). Conformément au diagramme d'utilisation, cette interface ne permet pas d'ordonner au contrôleur un formatage de la consigne. En effet, l'opération format n'est pas accessible depuis cette interface.

Les acteurs utilisés par le contrôleur sont modélisés comme des interfaces requises (flèches pointillées blanches). Le contrôleur requiert les librairies *Playerc_client, Playerc_position2d* et *Playerc_sonar.* Les opérations de ces librairies sont décrites comme des opérations statiques puisqu'elles sont codées nativement comme des fonctions C dans les librairies du simulateur. Elles sont donc des opérations des interfaces elles-mêmes.

En ce qui concerne la représentation comportementale, chaque opération est décrite par un diagramme d'activité. La figure 4 présente un fragment de l'activité de connexion (connect) implantée par le pilote. Ce diagramme d'activité spécifie les actions effectuées par le pilote pour s'interfacer avec le simulateur. La première action est un appel de la méthode *statique create.* Cette méthode est décrite dans l'interface *playerc_client.* Elle renvoie au pilote un e socket client. Les paramètres et la valeur de retour de cet appel d'opération sont décrits sous la forme de broche (pin UML) de données et données (" datastore "). A l'issue de cet appel, une socket client a été créé pour dialoguer avec le serveur *Player* (et donc avec le simulateur *Gazebo*). Il faut désormais que le pilote se connecte à cette socket. Pour cela, il demande explicitement la connexion par l'intermédiaire de la méthode statique *connect,* méthode fournie elle aussi dans l'interface *playerc_client.* Si cette connexion est réussie, l'activité de connexion est terminée. Sinon, le pilote émet une erreur à l'ensemble des classes du modèle UML. Cette émission est modélisée par l'intermédiaire d'un signal *Erreur.* Ce signal est géré par une classe spécifique non décrite dans la figure 3.

A l'issue de cette étape de conception, la structure de l'application et la description de chacune des opérations des classes *Controller* et *PlayercDriver* ont été définies. Même si ces descriptions apparaissent comme des diagrammes, les concepts auxquels ils font référence ont un sens au même titre que les mots d'un langage de programmation. Ainsi, ces diagrammes peuvent être utilisés par des outils logiciels destinés à la production (semi-)automatique de l'application.

Implantation de l'application

L'implantation (semi-)automatique de l'application passe par l'utilisation de générateurs. Des technologies comme Acceleo (3) permettent de décrire des générateurs de code. L'utilisateur peut décrire, à partir des concepts du langage UML (classes, interfaces et opérations par exemple), un générateur ciblant un

langage de programmation particulier. Dans cet article, le langage cible est le langage C. Un générateur de code C est donc utilisé. La communauté Acceleo fournit d'ailleurs, dans sa ferme de modules, un exemple opérationnel de générateur de code C. Pour cet article, ce générateur a été étendu pour gérer les diagrammes d'activité, les interfaces et les signaux.

La figure 5 illustre l'utilisation du langage Acceleo pour générer le code C de cette application.

De manière synthétique, chaque classe est transformée en un fichier source (.c) et un fichier d'en-tête (.h). Chaque classe produit une structure. Les méthodes sont transformées en des fonctions. Les interfaces requises (*Playerc_client, Playerc_ position2d et Playerc_sonar*) ne produisent aucun fichier puisque ce sont des imports de librairies existantes. L'interface fournie, *Icontroler,* produit un fichier source et un fichier d'en-tête contenant une fonction *updateControl.* Cette fonction redirige les appels à la fonction *updateControl* décrite dans le fichier *source Controller.*

Conclusion

Face à l'abondance des concepts et des notations que propose ce langage, ce dossier avait pour objectif d'initier l'utilisation des principaux diagrammes UML pour le développement d'une application logicielle. Pour cela, une application logicielle a été tout d'abord spécifiée par des diagrammes de cas d'utilisation. Ensuite, les diagrammes de classe et d'activité ont été illustrés pour la conception structurelle et la conception comportementale de cette application. Puisque modéliser n'est pas un but en soi, mais un moyen, les modèles associés à ces diagrammes ont été utilisés pour générer l'implantation de l'application, c'est-à-dire le code C de l'implantation. Bien sûr, à partir de ces mêmes modèles, des générateurs JAVA ou C++ auraient pu être utilisés pour cibler des implantations différentes du simulateur.

Outre l'utilisation de tel ou tel diagramme dans telle ou telle étape de développement, un point important à retenir est l'utilisation d'UML comme un langage et non comme une méthodologie. L'utilisation d'UML nécessite donc une méthodologie annexe permettant de contraindre l'utilisateur et donc de guider les modélisations pour que celles-ci soient utilisables par des outils informatiques. Ces méthodologies, ces langages et ces outils s'intègrent alors dans des approches de développement qui sont dites dirigées par les modèles. C'est ainsi que des technologies émergentes basées sur les modèles arrivent aujourd'hui à maturité. Les technologies EMF (4) et GMF (5) de la fondation Eclipse (6) permettent par exemple de générer automatiquement des éditeurs textuels et des éditeurs graphiques. Le modeleur Papyrus (7), utilisé dans cet article, est d'ailleurs en partie généré grâce à ces technologies.

3 - www.acceleo.org
4 - http://www.eclipse.org/modeling/emf/
5 - http://www.eclipse.org/modeling/gmf/
6 - http://www.eclipse.org/
7 - http://www.papyrusUML.org/

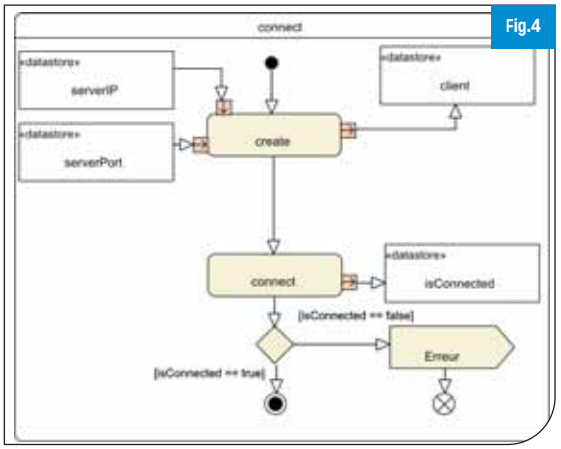


Figure 4 : Conception de l'activité de connexion du pilote

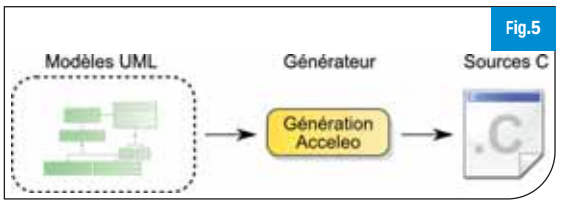


Figure 5 : Synoptique de génération de code

■ Frédéric **Thomas**
Docteur en ingénierie dirigée par les modèles et consultant de la société Obeo
http://www.obeo.fr

Parlons qualité et patrimoine des modèles

Le " legacy model "

Qu'entend-t-on par patrimoine ? Il s'agit en quelque sorte du " legacy model ", c'est-à-dire de l'ensemble des modèles utilisés par l'entreprise, le modèle depuis x années. En effet, nous faisons de la modélisation depuis des années et ces modèles il faut bien les préserver. Mais comment faire lorsque ces modèles sont de l'UML 1.x sur une ancienne version d'un modeleur ou encore sur un outil disparu ? Là est toute la question car avec un ancien UML, le format d'interopérabilité XMI ne garantit pas une bonne reprise du patrimoine ancien. Que faut- il faire ? Migrer les modèles de version en version d'outils et d'UML ? Refaire entièrement chaque modèle ? A l'heure actuelle, cette préoccupation n'existe pas ou peu. Et à notre connaissance les éditeurs ne proposent rien sur ce domaine.

La qualité des modèles

La qualité, la qualification des modèles constitue un autre défi pour les modeleurs. *" Il existe différentes approches sur ce domaine ",* avance **Guy Cartigny** de Compuware France. Aujourd'hui, les spécifications (=expression des besoins) d'une application permettent de générer tout ou partie d'un modèle, ce qui peut garantir un minimum la qualité du modèle, si les besoins sont structurés et bien exprimés. *" Mais si le modèle est complexe et peu clair, l'application générée le sera aussi "* continue M. Cartigny. La qualité d'un modèle va aussi dépendre de sa granularité. Il faudrait un compromis, quitte à scinder en plusieurs modèles les spécifications exprimées. On peut utiliser des métriques de type SDmetrics mais il ne faut pas en être prisonnier car elles ne sont qu'un élément de qualité. La qualité d'un modèle viendra notamment de sa concision, sa clarté, sa précision.

■ François **Tonic**

Générer du code avec UML en quelques clics !

Dans cet article, nous nous intéresserons à la génération de code à partir d'UML. Nous verrons quels sont les diagrammes exploitables et quelles en sont les bonnes pratiques. Tous les exemples que nous proposons ici sont illustrés en Java mais il est important de souligner que n'importe quel autre langage de programmation peut y être substitué.

UML propose 13 types de diagrammes permettant de spécifier les différentes facettes d'une application. Sur ces 13 types de diagrammes seuls 3 sont directement exploitables pour la génération de code : le diagramme de classes, le diagramme de machine à états et le diagramme d'activité.

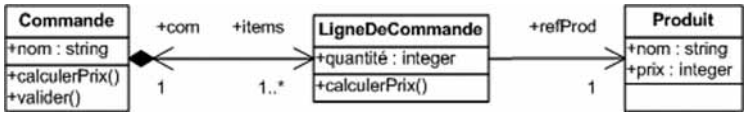
Les diagrammes (utiles)

Le diagramme de classes permet de spécifier les différentes classes qui composent une application. A partir d'un diagramme de classes, trois parties d'un code peuvent être générées : le **squelette** de code, les **accesseurs** et les **contrôles d'intégrité.** Le **squelette** de code correspond à la définition des classes, de leurs attributs et des signatures de leurs méthodes. Cette génération ne contient pas de difficultés majeures et tout générateur digne de ce nom doit savoir gérer, entre autres, l'héritage de classes, la réalisation des interfaces ainsi que la conversion des types UML. Les **accesseurs** sont les méthodes classiques permettant de lire ou d'affecter les valeurs des attributs et des références. Cette génération n'est pas non plus complexe et il est souhaitable que le générateur de code propose plusieurs options de génération des accesseurs (offrir des méthodes permettant par exemple l'insertion d'un nouvel élément dans une liste ou la suppression d'un élément existant). **Les contrôles d'intégrité** sont quant à eux très complexes à mettre en œuvre et rares sont les générateurs de code qui les supportent. Les contrôles d'intégrité concernent principalement les associations UML et assurent que leur sémantique se retrouve dans le code.

A titre d'exemple, considérons le diagramme de classes présenté dans la figure 1 qui spécifie qu'une commande est composée de plusieurs lignes de commande (au moins deux) et que celles-ci référencent chacune un produit. L'agrégation (diamant noir) précise que si une commande est détruite alors toutes ses lignes de commandes doivent être détruites.

Le code suivant présente une génération de code vers Java pour la classe Commande. Le squelette de code correspond à la définition de la classe, de ses attributs et des signatures de ses méthodes. Le code des accesseurs est assez classique (insertion d'un élément dans une liste). Le code de contrôle d'intégrité est quant à lui bien plus complexe. La méthode *itemsLowerBound* permet uniquement de savoir si le nombre minimal de lignes de commandes est respecté (l'agrégation n'est pas ici supportée). Cet exemple permet de mieux comprendre pourquoi la plupart des outils s'arrêtent à la génération de squelette de code.

Fig.1




```
import java.util.List;
import java.util.ArrayList;
public class Commande {
    //squelette
     public List<LigneDeCommande> items = new Array
List<LigneDeCommande> ();
    public String nom;
    public int calculerPrix()    {//TODO }
    //Accesseurs
     public void insertLigneDeCommandeAt(LigneDeCom
mande l , int pos) {items.add(pos,l) ;}
    //Contrôle d'intégrité
     public boolean itemsLowerBound() {return items
.size()<2 ;}
}
```

Le deuxième diagramme UML qui peut être utilisé pour générer le code est **le diagramme de machine à états.** Un diagramme de machine à états permet de spécifier les différents états que peut avoir un objet instance d'une classe. Il permet entre autres de spécifier les suites valides d'appels de méthodes et celles qui sont interdites. La génération de code à partir d'une machine à état est un domaine qui a été fortement étudié ; le pattern **State** du *Gang of Four* lui est même entièrement dévolu. Pour autant, les machines à états UML2 sont tellement complexes que rares sont les outils qui offrent une génération complète de code à partir de celles-ci.

A titre d'exemple, considérons la machine à états de la figure 2. Celle-ci spécifie qu'une fois une commande validée, il n'est alors plus possible d'y ajouter de nouvelles lignes. Une génération de code devrait permettre par

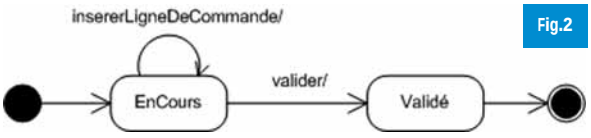


Fig.2

exemple d'ajouter un test à la méthode *insererLigneDeCommande* afin de s'assurer que la commande n'a pas été validée avant l'insertion d'une nouvelle ligne de commande.

```
public void insertLigneDeCommandeAt(LigneDeCom
mande l , int pos) {if (!valider) items.add(pos,l) ;}
```

Enfin, le troisième diagramme exploitable pour la génération de code est **le diagramme d'activité**. Une activité spécifie quelles sont les actions réalisées dans un comportement en précisant leur ordre d'apparition ainsi que les données manipulées en entrée et en sortie. Les diagrammes d'activité UML2 peuvent être attachés à une méthode pour spécifier l'intégralité de son corps. Ces diagrammes sont tellement expressifs qu'une génération de code pourrait être mise en œuvre sans trop de complexité. Pour autant, leur utilisation à ce niveau reste quasi inexistante. Le problème majeur est qu'il n'existe pas de notation standard suffisamment concise permettant leur utilisation. Par exemple, spécifier le corps de l'opération *calculerPrix* de la classe *Commande* nécessiterait un diagramme composé d'au moins une vingtaine d'éléments tenant difficilement sur une page A4. Ceci explique pourquoi il n'existe aucun outil du marché offrant par défaut une telle capacité de génération de code.

Etes-vous Round Trip ?

Actuellement deux approches de conception guidée par les modèles sont identifiées : l'approche **Model Driven** et l'approche **Round Trip.** Nous aborderons ici l'approche round trip. L'approche **RoundTrip** consiste à effectuer des modifications dans le code et dans le modèle et à assurer une cohérence grâce aux opérations de génération de code et de reverse engineering. Cette approche bien que moins productive est assez appréciée par les développeurs car ils peuvent continuer à utiliser leur outil de développement et toutes les facilités qu'ils offrent (debug, refactoring, etc.). La plus grande difficulté de cette approche consiste à maintenir la cohérence entre le code et le modèle. Actuellement, la solution supportée par de nombreux outils est d'intégrer dans le code des marqueurs délimitant les zones de code qui ne doivent absolument pas être modifiées sous peine de briser la cohérence.

Les bonnes pratiques

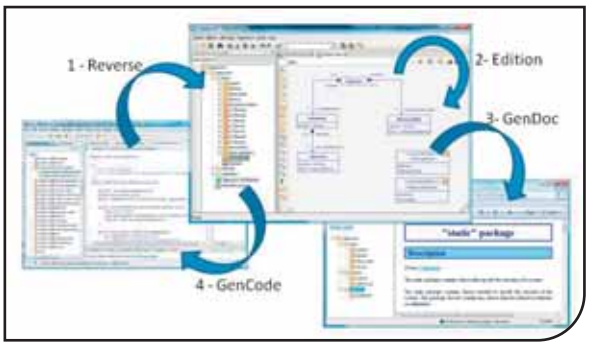
L'utilisation d'UML pour générer du code nécessite de bien connaître l'outil de modélisation que l'on utilise et surtout de connaître les opérations de génération de code et de reverse engineering qu'il offre. En effet, si UML est un langage standard, les opérations de production sur UML ne sont quant à elles pas standard. La génération de code UML vers Java dépend donc énormément de l'outil qu'on utilise. Sachez choisir et utiliser l'outil qui offre les services dont vous avez besoin. Si vous avez choisi de faire du round trip, vérifiez bien que votre outil supporte cette approche et facilite la cohérence entre votre modèle et votre code. Sachez utiliser les diagrammes de machines à états et les diagrammes d'activité avec parcimonie si votre objectif premier est la génération de code. En effet, ces diagrammes ne sont exploitables que selon l'approche Model Driven car aucune opération de reverse engineering n'existe aujourd'hui pour eux.

N'utilisez pas les diagrammes de séquence pour générer du code. Ceux-ci ne sont pas directement exploitables. En effet, un diagramme de séquence spécifie un exemple d'échange de messages entre différents objets. Pour pouvoir être exploité et permettre une génération de code, il faudrait avoir plus d'information et savoir par exemple si le diagramme de séquence arrive tout le temps et ce, quel que soit l'état des objets.

Dans vos modèles UML, séparez les parties qui vont vous servir à générer du code de celles qui vous servent aux phases amont (gestion des exigences et phase d'analyse).

Mise en pratique

Pour démarrer l'utilisation d'UML, je vous conseille de prendre une application existante et d'effectuer un reverse engineering. Vous obtiendrez alors un diagramme de classes représentant exactement votre application et pourrez générer automatiquement une documentation de votre conception. De plus, vous pourrez modifier graphiquement votre conception. Rien ne sera alors plus facile que d'appliquer des patterns ou de changer la découpe en package. En utilisant la génération de code vous obtiendrez alors directement le nouveau code de votre application.



Dans mon cours à Jussieu je propose à mes étudiants de suivre cette approche. En moins de 10h de cours, ceux-ci deviennent des experts et utilisent Objecteering pour réaliser et documenter la conception de toutes leurs applications Java. N'hésitez pas à lire et à relire notre livre sur le sujet " UML2 pour les développeurs " chez Eyrolles.

■ Xavier **Blanc**
Maître de conférences à l'Université Pierre et Marie Curie (Paris 6), responsable de l'enseignement des techniques de modélisation en 3e année de licence et en master d'informatique. Ses recherches portent sur l'ingénierie guidée par les modèles. Représentant du LIP6 (Laboratoire d'informatique de l'Université Paris 6) à l'OMG. Auteur de MDA en action et UML2 pour les développeurs (Eyrolles)

La modélisation au cœur des applications métier

Leonardi, édité par Lyria (racheté par l'éditeur W4) s'inscrit clairement dans la mouvance de l'ingénierie pilotée par les modèles (Model Driven, ou MD). Dans cet article, nous définissons Leonardi avant d'expliquer ce qu'est son modèle métier et nous concluons en citant les principaux avantages de l'approche mise en œuvre.

Le principe de fonctionnement

Leonardi est un environnement pour produire des applications métier avec Interfaces Homme Machine (IHM) évoluées. Son approche innovante basée sur le modèle métier permet d'automatiser en grande partie la chaîne de production intervenant classiquement dans les IHM. L'outil se distingue des outils traditionnels de type MDA car il ne génère pas de code : c'est en effet son moteur d'exécution qui interprète à la volée le modèle métier, générant dynamiquement l'interface utilisateur.

Le modèle métier

L'édifice central du framework est donc le modèle métier, formalisé par des fichiers XML et élaboré de manière itérative pendant la mise au point de l'application. Mais que contient ce modèle ? Avant tout, la description objet des données manipulées dans le projet par l'utilisateur final. Cela se traduit par un ensemble de classes métier structurées par des attributs typés (textes, nombres, dates, relations, énumérations...) et agrémentées par des actions disponibles pour l'utilisateur final : édition, création, impression, import / export... Cette structure représente une réelle abstraction fonctionnelle du monde métier, qui est ensuite complétée par des informations de nature diverse.

Pour **l'IHM,** cela inclut les labels à afficher, pris dans des dictionnaires en fonction de la langue, mais aussi :

- L'**arbre de navigation,** qui formalise les enchaînements possibles entre écrans. Cela comprend l'action racine du projet (typiquement un login) et une hiérarchie d'actions pour accéder aux données, à des profondeurs diverses.
- Les **processus métier,** qui synchronisent l'affichage en intégrant la notion de workflow collaboratif, pour orchestrer l'apparition du bon écran au bon moment pour le bon interlocuteur.
- Au niveau des **classes,** un filtre peut limiter les données à charger et la politique de **cache** à utiliser lors du chargement des objets en mémoire (au démarrage, lors de la demande d'accès, etc.) est indiquée.
- Au niveau des **attributs,** des marques spécifient le comportement attendu à l'exécution: utilisation pour le nommage, présence dans les formulaires (consultation, édition, création) ou dans les tables (pour le filtrage, le tri...), caractère optionnel ou obligatoire, disposition du champ dans le formulaire, liens avec d'autres champs dans un onglet ou un groupe... Des règles peuvent décrire les comportements des formulaires (disponibilité lors de la saisie en fonction de valeurs d'autres champs) et des formats de saisie peuvent être imposés sous forme d'expressions régulières.
- Au niveau des **actions,** le modèle contient notamment les conditions de disponibilité en fonction de la sélection en cours dans la vue, du contexte d'accès ou des valeurs d'attributs.

Pour la **persistance des données,** des références vers la couche physique permettent d'indiquer la provenance des données alimentant les écrans. Une classe métier peut ainsi donner une vue homogène de données hétérogènes issues de différentes sources (fichiers, BD, LDAP, workflow, Corba, GED, formats propriétaires, etc.). Enfin, lorsque cela s'avère utile, le modèle contient aussi des pointeurs sur des **classes Java** qui prennent le relais, à l'exécution, des classes déjà embarquées dans le moteur. Ce code spécialise le comportement des classes génériques (associé à un composant graphique, la session, une classe métier, etc.). Il est généralement utile pour doter l'application de véritables règles fonctionnelles.

Conclusion

Cet aperçu du modèle métier au sens Leonardi éclaire sur la richesse et la flexibilité que procure l'approche MD lors de la mise au point des applications métier. L'approche MD permet d'abord d'accroître spectaculairement la productivité. Elle permet ensuite de séparer la technologie du métier : les experts business tiennent ainsi un rôle prépondérant (par rapport aux experts techniques) et expriment directement leur besoin dans le modèle. Le modèle peut être élaboré avec l'atelier Studio de Leonardi. Il peut aussi être découvert en introspectant une BD existante ou à partir d'un modèle UML, et enrichi par la suite. Le moteur (alternative à la génération de code) confère une grande agilité pendant tout le cycle de vie du logiciel et les résultats sont visibles rapidement. Ce fonctionnement permet en outre de s'affranchir de la problématique classique de synchronisation entre le code et le modèle : lorsque ce dernier évolue, l'application reste à niveau, sans effort.

■ Jean-Loup **Comeliau**
Lyria (groupe W4) - jean-loup.comeliau@lyria.com

Quel modèle pour quel usage ?

Modéliser, c'est représenter de manière abstraite un problème ou une solution, afin de pouvoir raisonner sur ceux-ci. Ainsi, deux grandes situations d'utilisation de modèles existent :

1. Je cherche à comprendre un problème : il s'agit là d'une activité d'*analyse,* où il est nécessaire de répertorier l'ensemble des données du problème, de les structurer, de les formaliser, et de réduire leur complexité à ce qui est utile et nécessaire.
2. Je cherche à décrire une solution : il s'agit là d'une activité de *conception.* Il faut décrire le système prévu, analyser son fonctionnement, ses modes d'utilisation, et étudier sa mise en place et son exploitation sous différents aspects comme par exemple ses tests, son administration, sa maintenance, etc.

On voit ici que l'activité de modélisation dépasse largement le cadre du logiciel. Il existe des modèles pour toutes sortes de domaines : le bâtiment, la chimie, la biologie, la physique, etc. Le logiciel lui-même n'est souvent qu'un constituant d'un système plus large comme typiquement une entreprise incluant son système d'information ou un système technique - (avion, système de contrôle aérien, ...) où le logiciel interagit avec du hardware (ordinateur bien sûr, mais aussi radar, GPS, ...), du matériel (fuselage d'avion, radar, système de commande mécanique, ...) et des interventions humaines. Ces deux exemples très importants pour le logiciel font l'objet de disciplines de modélisation dédiées, comme *l'ingénierie des systèmes* avec le standard SysML (extension de UML) pour les systèmes techniques, et *l'Architecture d'Entreprise* incluant, par exemple, les disciplines nommées cartographie et urbanisation en France.

Application des modèles au logiciel

Pour le logiciel, modéliser ne s'impose pas dans toutes les situations : par exemple, rédiger quelques macros Excel ou réaliser une page web ne nécessite bien souvent pas l'emploi de cette pratique. Si l'emploi de modèles est indispensable dans les situations ou l'application atteint un certain volume, et où le nombre d'intervenants est important, il s'avère cependant extrêmement utile très tôt : par exemple, un modèle conceptuel peut comprendre seulement 3 classes, mais apporter une grande plus-value en positionnant correctement les concepts, en évitant les redondances, en permettant de simuler (au moins mentalement) ses différentes manipulations et en matérialisant une définition partagée entre les intervenants. Vouloir organiser quelques tables en base de données, des tableaux Excel ou un outil de CRM ; vouloir comprendre un processus, avec les responsabilités de chacun, ce qu'il y a à faire et pourquoi il y a des problèmes récurrents sur sa bonne réalisation - sont deux cas très simples où un modèle apportera immédiatement sa plus-value (vision abstraite du problème, structuration, redondances évitées, vision partagée, simulation de scénarios, ...). Le modèle rend explicite un problème ou une solution : c'est là sa qualité première.

Le logiciel pose le problème aigu de la maîtrise du code développé : Comment guider son développement coordonné dans une grande équipe et sur un gros volume ? Comment faciliter sa maintenance, ou comment savoir ce qu'il y a au sein de centaines de milliers de lignes de code, à quoi elles servent et où intervenir ? Par sa capacité d'abstraction, le modèle est " n " fois plus compact que le code. Selon son niveau d'abstraction (variable selon son objectif, le point de vue considéré sur le système), il peut typiquement abstraire à un facteur 10, 100, 1 000, 10 000.

Bien sûr, ceci reporte la question sur le modèle : que vaut-il ? Comment garantir sa maîtrise ? Mais le problème est reporté sur des volumes plus faciles à maîtriser et à abstraire. In fine, la démarche, la méthode seront les garants de la qualité. Ce raisonnement conduit mécaniquement au développement guidé par le modèle (approche MDA – Model Driven Architecture) : il faut que le modèle représente effectivement le code, il faut aussi qu'il n'y ait pas double travail – modélisation et codage. Les outils supportant la génération de code d'après le modèle, comme l'atelier Objecteering évitent le travail redondant, et maintiennent la cohérence code modèle. L'argument de la productivité, pourtant largement utilisé, est vrai, mais est secondaire face à l'enjeu de maîtrise des développements logiciels.

Quel modèle utiliser ?

A travers UML et BPMN, les standards nous fournissent une boîte à outil couvrant une large partie des besoins de modélisation logicielle. Il reste des parties à couvrir par des extensions, comme par exemple SysML pour les systèmes techniques, qui est une extension d'UML (Profile). D'autres domaines comme l'Architecture d'Entreprise n'ont pas encore de standards établis (l'OMG y travaille cependant). Des notations propriétaires sont supportées

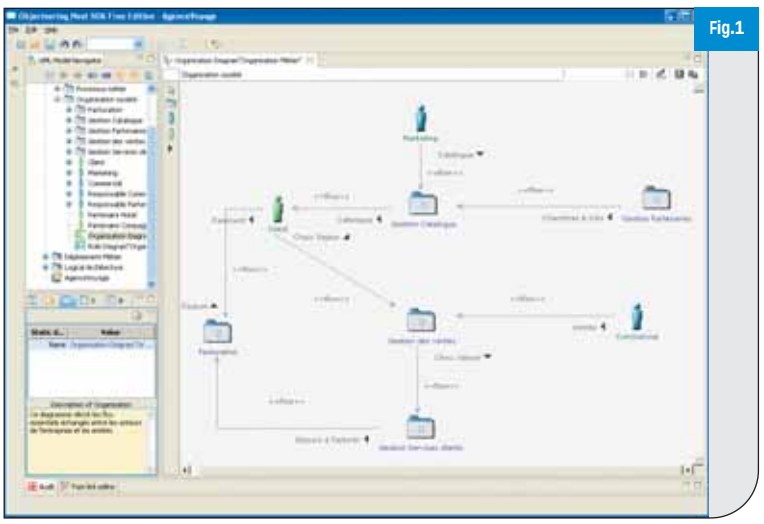


Fig.1

Diagramme d'organisation (Adaptation UML pour l'Architecture d'Entreprise).

par différents éditeurs. En prévision des résultats des travaux OMG et en s'appuyant sur UML, l'atelier Objecteering (www.objecteering.fr) fournit par exemple des extensions UML pour supporter la modélisation d'architecture d'entreprise, intégrée avec les standards UML et BPMN. Enfin, les domaines plus en amont, comme par exemple l'analyse des exigences ou encore l'analyse des objectifs peuvent être aussi supportés par la modélisation. SysML apporte un standard pour l'analyse des exigences. L'analyse des objectifs, ainsi que la modélisation des règles métier sont supportés par des modèles propres non standardisés. [Fig.1]

Comme le résume le tableau, chaque niveau, chaque étape d'analyse, conception ou réalisation de systèmes dispose ainsi de modèles adaptés. Il reste à déterminer la plus value de chaque modèle dans chaque situation, et la bonne façon de les mettre en œuvre. C'est là une affaire de méthode et d'expérience. Une bonne façon de l'assimiler est de pratiquer, pourquoi pas en utilisant la version gratuitement téléchargeable de l'atelier Objecteering.

Type d'utilisation	Modèle proposé
Analyse des besoins	SysML, modèles spécifiques
Analyse des Objectifs	Modèles spécifiques
Architecture d'entreprise Modélisation	Modèles spécifiques, Extensions UML (Profiles)
des processus métier	BPMN, Diagrammes d'Activité UML
Systèmes techniques	SysML
Cas d'utilisation : analyse logiciel	Use Case UML
Règles métier	Modèles spécifiques
Organisation de système	Diagrammes de packages, diagrammes de Flux (UML) – Modèles spécifiques
Modèles conceptuels	Diagrammes de classe & Package (UML)
Modèles de données	Diagrammes de classe & Package (UML)
Modèle Objet, Modèles pour développement (Java, C#, ...)	Diagrammes de classe & Package (UML)
Modélisation de scénario (analyse, conception, tests, simulation)	Diagramme de séquence (UML) diagramme d'activité
Design pattern	Diagrammes de collaboration(UML)
Architecture technique	Diagrammes de classes structurées ou composants (UML)
Déploiement sur matériel	Diagrammes de déploiement (UML)
Dynamique d'exécution des objets	Diagramme d'état, Diagrammes d'activité

Productivité des modèles pour la programmation

La modélisation UML pour le développement logiciel est désormais une pratique très répandue dans les équipes de développement. Cependant, les scénarios varient beaucoup selon les contextes : Depuis l'absence de modèles, en passant par l'utilisation d'ateliers UML léger open source par des développeurs isolés, jusqu'à la mise en place d'ateliers plus sophistiqués pour les équipes appliquant un processus de développement et une démarche qualité.

Quel gain de productivité peut-on en attendre ? Une première indication de productivité réside dans le pouvoir d'abstraction d'un langage par rapport à un autre : le nombre d'éléments produits dans le langage cible pour un élément dans le langage d'origine.

Ainsi, la productivité immédiatement perçue des ateliers UML est souvent vue en termes de nombre de lignes de code créées par élément de modèle UML.

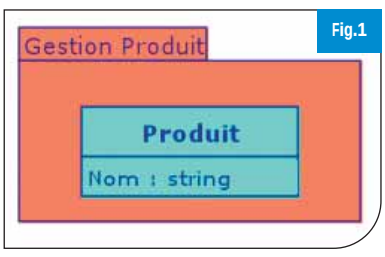
Dans l'exemple ci-dessous (UML –> Java), la productivité perçue est faible : on réduit UML à de la programmation visuelle. [Fig.1]

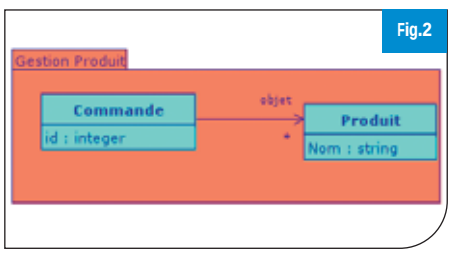
```
package Gestion_Produit;

public class Produit {
     public String Nom;
     public String getNom () {
          return this.Nom;
     }
}
```

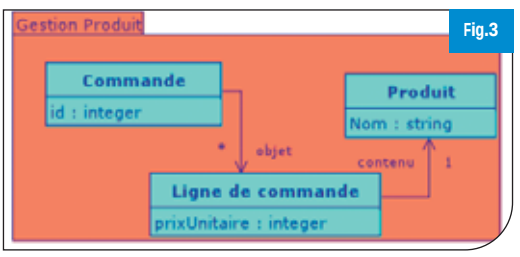
L'emploi de notions évoluées comme les associations produit plus de code, et illustre une mise en œuvre plus sophistiquée du langage (ici, les templates). [Fig.2]

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

public class Commande {
     public int id;
     public int getId () {
          return this.id;
     }
    public List<Produit> objet = new ArrayList<Produit>();
     public List<Produit> getObjet () {
          return this.objet;
     }
}
```

Lors de retouches du modèle, la perception de la productivité augmente. Ici, le drag & drop de l'extrémité de l'as-







sociation (rôle " objet ") sur une autre classe produit de nombreux changements dans le code généré. [Fig.3]

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

public class Commande {
    public int id;
    public int getId () {
        return this.id;
    }
    public List<Ligne_de_commande> objet =
new ArrayList<Ligne_de_commande> ();
    public List<Ligne_de_commande> getObjet () {
        return this.objet;
    }
}
```

Mais la productivité se mesure aussi sur la durée d'une application : il est ainsi très facile de réaliser rapidement un grand nombre de lignes de code avec un IDE. On assiste émerveillé à une forte productivité d'un développeur, mais les étapes suivantes peuvent entraîner des désillusions quand l'application réalisée ne correspond pas aux attentes, quand une autre personne a du mal à reprendre le code existant, c'est-à-dire aux étapes révélatrices d'intégration, de validation et de maintenance. Trop souvent, on se limite à une productivité perçue, correspondant au temps mis pour réaliser les premières fonctions d'une application, sans considérer la productivité dans son ensemble. Les facteurs d'échecs et retards dans les projets sont essentiellement liés à l'imprécision des spécifications, à la mauvaise communication entre équipes liées au métier et au besoin avec les équipes de réalisation. D'autre part, plus de 70% du coût investi dans une application correspond aux coûts liés à sa maintenance. Dans ce coût, outre la qualité du code produit, intervient la facilité de compréhension du code, de reprise et d'évolution.

C'est sur ces points que l'approche de développement guidée par le modèle apporte une plus value importante : En reprenant des modèles issus d'étapes plus en amont d'analyse, le développeur assure une traçabilité renforcée entre les travaux d'analyse et ceux de réalisation. Certes, le modèle doit être retouché pour s'appliquer au code, et il n'y a pas de correspondance totale modèle d'analyse / modèle de code, mais on a une bien meilleure traçabilité, une reprise des notions et de la terminologie, et un dialogue facilité. Les ateliers les plus sophistiqués offrent des mécanismes permettant par ailleurs de gérer cette traçabilité.Nous verrons que les aspects " systématiques " d'une génération de code permettent à une équipe de partager des conventions bien établies, très utiles notamment en maintenance.

La génération de code, jusqu'où ?

Dans la qualité d'une application produite intervient l'expertise résidant dans les règles de transformation modèle / code : la manière optimale de traduire les formes de modélisation en code, de bien exploiter l'infrastructure ciblée, est automatisée, systématisée, et garantie correcte sur toute l'application. Bien souvent, de nouvelles optimisations des règles sont trouvées lors du déroulement du projet : dans ce cas, les règles de transformation sont mises à jour, et l'ensemble du code existant est alors reprogrammé. A ce stade, intervient la faculté de paramétrage de la génération de code, qui doit être complète. La plus value du générateur est d'autant plus forte que le langage est complexe : par exemple, le gain avec C++ peut être considérable, en fournissant une disposition des " .hxx " et " .cxx " homogène, une règle de production des " include " optimisée en fonction du modèle, une gestion fine des pointeurs et containers, etc.

Certains générateurs se limitent à la génération de squelettes à partir du modèle : leur plus value est faible, et leur usage est rapidement abandonné dans un projet. Sur la seule base du modèle statique, certaines applications produisent jusqu'à 80% du code : celui-ci prend en charge les parties déclaratives, la persistance, des mécanismes liés aux frameworks employés, la liaison MVC avec l'interface homme / machine, etc. Il y a par exemple des générateurs pour Hibernate, pour les EJB, pour Spring, ou pour Struts (en exploitant les diagrammes d'activité).

UML2, avec le support des composants permet de produire des patterns isolants dans les composants logiciels, et assurant un véritable " plug & play " de ceux-ci.

Les générateurs les plus sophistiqués produisent également la chaîne de production de code à partir du modèle, en s'appuyant sur le modèle de déploiement UML : l'enfer des makefiles, de fichiers ANT ou Maven, toujours difficiles à maintenir, rarement documentés devient ainsi modélisé, et simplement produit.

La génération de 100% du code à partir du modèle est possible. Elle se produit en utilisant les modèles dynamiques (diagrammes de séquence UML, diagrammes d'activité, diagrammes d'état, " action semantics "). En pratique, elle adresse surtout des domaines spécialisés comme l'embarqué ou le temps réel.

Très rapidement, sur les grandes applications, le besoin d'une approche MDA (Model Driven Architecture) se fait sentir : Il s'agit souvent d'adapter la génération de code à des frameworks, pour que la génération de code exploite les modèles pour appliquer les règles d'architecture et génère pour ceux-ci. Ici, la productivité est maximisée, en même temps que la complexité est réduite, car les mécanismes techniques de " mapping " modèle / framework sont systématiquement appliqués.

De manière logique, la productivité et la qualité sont maximisées par une combinaison adaptée entre IDE, Frameworks et ateliers UML, ce qui nécessite des ateliers de haut niveau, capables de combiner des générateurs de code de qualité, un support intégré des équipes, une bonne intégration aux IDE, et un support MDA évolué.

■ Philippe **Desfray** - *SOFTEAM*

MDSD, MDD ou MDA : quel outil choisir ?

Les enjeux des chaînes de Développement Logiciel Guidé par des Modèles (Model Driven [Software] Development – MDSD / MDD) résident dans une réduction des charges de codage autant que dans la garantie de la qualité du code produit.

Les gains, réels et pragmatiques, constatés sur les projets bénéficiant de ces approches sont de 10 à 20% en réalisation initiale et de 30 à 50% en maintenance. Encore faut-il pour cela avoir retenu un ensemble d'outils adaptés à son contexte et à ses besoins. Cinq questions clefs permettent de bien cadrer les offres du marché en la matière.

COMPRENDRE LES OFFRES MDSD EN 5 QUESTIONS

Quelles transformations ?

La chaîne MDSD a pour rôle de créer des codes sources à partir de modèles qui lui sont fournis en entrée. Ces modèles sont autant que possible indépendants de la solution technique cible retenue. Ce sont des Modèles Indépendants de la Plate-forme (PIM) dans la terminologie MDA (Model Driven Architecture), l'approche MDSD formalisée par l'OMG (1).La capacité de la chaîne MDSD à accepter en entrée différents types de modèles répondant à différents formalismes graphiques ou textuels est essentielle. Il s'agit, d'une part, de modèles exprimés dans un formalisme normalisé ou standardisé, comme UML, d'autre part, de modèles exprimés dans un formalisme spécifique à un domaine (Domain Specific Model - DSM).Pour exploiter ces modèles, la chaîne MDSD s'appuie sur deux catégories de transformations : les transformations modèles à texte (M2T) et les transformations modèles à modèles (M2M).Les transformations M2T sont dédiées à la production du code source (Java, C#, HTML, XML,...) à partir de modèles. Elles produisent les fichiers qui, en l'état ou moyennant compléments des développeurs, sont compilés ou assemblés pour construire l'application cible.

Les transformations M2M permettent de produire un ou plusieurs modèles à partir d'un ou plusieurs autres modèles. Ces transformations peuvent être chaînées. Une première transformation M2M peut par exemple être en charge des spécificités de l'architecture logique (services métiers, couche d'accès aux données,...). La transformation suivante pourra quant à elle, par exemple, introduire dans les modèles les spécificités techniques de la cible (JEE, .Net, PHP, Spring, JPA, ...). Les modèles obtenus à l'issue de cette transformation sont, dans la terminologie MDA, des Modèles Spécifiques à la Plate-forme (PSM).

Cette décomposition de la chaîne MDSD en plusieurs transformations apporte à la chaîne une grande flexibilité et une grande adaptabilité aux évolutions de la plate-forme cible (évolution de normes, changement de solution technique,...).

La figure 1 illustre les liens entre les différents types de modèles introduits ci-dessus et les transformations qui les lient.

Quel langage de transformation ?

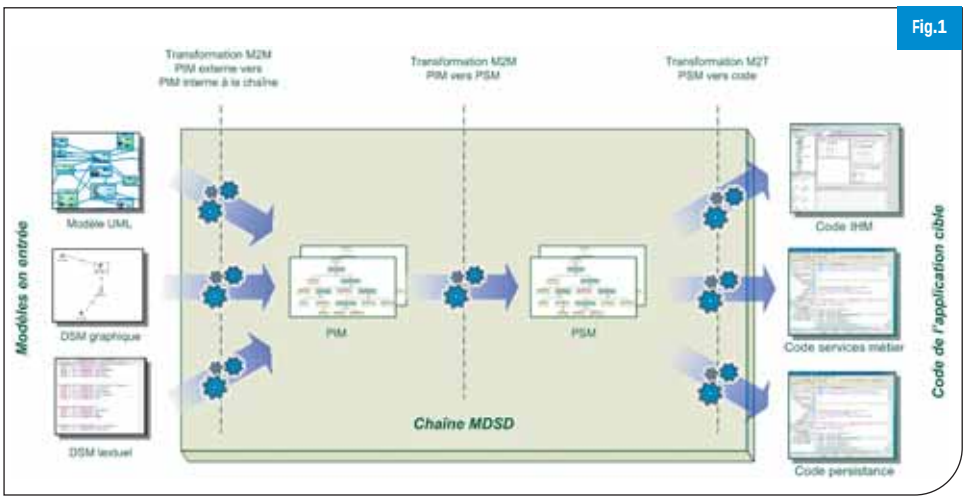
Les outils de transformations M2T sont des générateurs de code. Ils exploitent des moteurs de templates. Le rôle de ces moteurs est de générer du code à partir, d'un côté, d'informations issues des modèles en entrée, de l'autre, de squelettes de code (" templates ") paramétrés. La partie dynamique des squelettes de code est décrite à l'aide d'un langage ad hoc, tel que le VTL (Velocity Template Language) pour le moteur Velocity.

Les outils du marché s'appuient soit sur des moteurs de templates propriétaires, open source ou non, soit sur des moteurs de templates standardisés par le marché, notamment Velocity, de la communauté Apache, ou JET (2) de la communauté Eclipse.

Les transformations M2M sont quant à elles assurées par des technologies

Illustration d'une chaîne MDSD

Fig.1



qui restent encore jeunes. Les moteurs de transformations que l'on retrouve dans les outils sont tous propriétaires. Ils exploitent des langages de transformations qui peuvent être propriétaires, ouverts tel ATL (3) ou normalisés tel QVT (4). Le choix des technologies de transformation supportées par l'outil MDSD a un impact sur la gestion des compétences des architectes responsables de la chaîne MDSD et sur la portabilité et la pérennité des transformations réalisées.

Quelle approche de génération ?

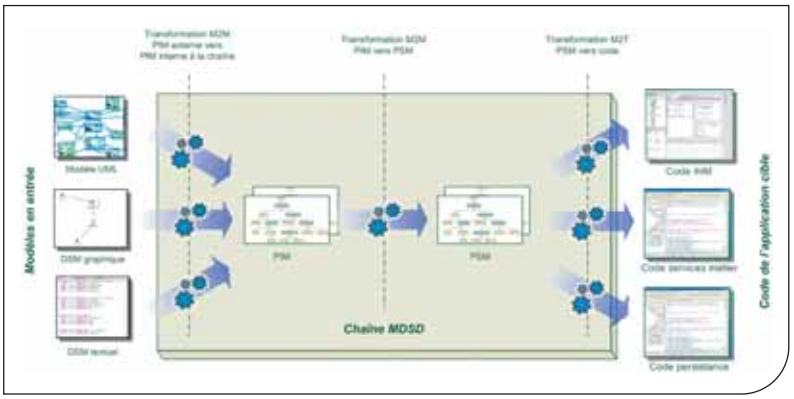
De façon générale, générer du texte, donc du code, est particulièrement aisé avec les technologies actuelles. Les difficultés résident dans le système de templating et dans la gestion de la génération incrémentale du code source. La qualité du système de templating détermine la facilité que les architectes ont à créer et à paramétrer les squelettes de code utilisés par le moteur de génération de code. La gestion incrémentale du code permet de générer à volonté différentes parties de la cible sans écraser les modifications effectuées manuellement par les utilisateurs.

La prise en compte du code des utilisateurs reste un des grands enjeux techniques des offres de génération de code. Face à ce constat, deux approches s'offrent à l'architecte. La première s'appuie sur la sélection d'une solution de génération ne sachant pas gérer le code utilisateur (solutions basées sur le moteur de templates Velocity par exemple) complété par une stratégie de génération de code intrusive. Cette stratégie nécessite la production de code respectant des motifs de conception (design pattern) qui permettent de protéger le code utilisateur. Elle est limitée aux technologies cibles supportant une forme de surcharge (redéfinition de méthodes héritées en langages objets) afin de permettre la génération d'une partie abstraite qui contient le code généré et d'une partie concrète, dite utilisateur, dans laquelle le développeur peut insérer son propre code. Cela exclut de facto les cibles telles que les fichiers HTML par exemple. Cette approche impacte la structure du code cible et peut en rendre la lecture plus complexe. Elle réduit cependant très fortement l'adhérence du code produit avec l'outil producteur et permet donc de faire évoluer ce dernier tout en maintenant la séparation code généré / code utilisateur.

La seconde s'appuie sur la sélection d'un outil gérant lui-même le contrôle du code utilisateur. Deux stratégies sont retenues par ces outils. La première consiste à placer des marqueurs dans le code. Ceux ci définissent des zones dans lesquelles l'utilisateur peut placer son code (JET, Acceleo, Xpand2). La seconde revient à maintenir un référentiel au sein de l'outil qui, pour chaque fichier, mémorise les modifications de l'utilisateur (Acceleo). Cette approche libère le code cible de toute considération liée à la manière dont il est produit (hors les marqueurs). Elle crée cependant une très forte adhérence du code produit avec l'outil producteur, ce dernier étant le seul à savoir quelles parties du code sont générées et quelles parties viennent de l'utilisateur.

Quelle intégration dans l'environnement de réalisation ?

Un outil de génération de code ne fonctionne jamais seul. Il est associé en amont avec un modeleur, en aval avec un environnement de développement. En la matière vous trouverez sur le marché des offres centrées sur un modeleur et intégrant les fonctions nécessaires de génération de code et des offres pure player MDSD centrées sur le générateur. La plupart de ces outils sont nativement intégrés ou intégrables à un environnement de développement du marché. En termes d'ingénierie logicielle, il est couramment attendu des concepteurs et ou des développeurs qu'ils pilotent les générateurs de code depuis des interfaces graphiques dédiées et produisent de fait automatiquement du code immédiatement disponible dans leur environnement de développement. Certains outils de génération ne disposent pas d'interface graphique. Leur déclenchement passe par la ligne de commande et donc, en pratique, par des scripts d'exécution. L'exécution par ligne de commande est certes moins ergonomique ; elle offre cependant l'opportunité d'insérer la chaîne MDSD dans une chaîne de construction et de packaging de l'application cible (chaîne Maven par exemple). De fait, la régé-



(1) OMG : Object Management Group, l'organisme international de normalisation des technologies objet.
(2) JET et Xpand : moteurs de templates du projet Eclipse M2T : http://www.eclipse.orsg/modeling/m2t/
(3) ATL : ATLAS Transformation Language, http://www.eclipse.org/m2m/atl/
(4) QVT : Query/View/Transformation, normalisé par l'OMG, http://www.omg.org/spec/QVT/1.0/

Le choix d'Auchan

Programmez ! : M. Lipka, pouvez vous nous dire deux mots sur votre société, sa DSI et votre fonction au sein de celle-ci ?

M. Lipka : Je suis Responsable de l'équipe Expertise & Outils dans le Département Ingénierie de développement. Ce Département est rattaché à la Direction Urbanisation & Service Transverse au sein de la Direction des Systèmes d'Information et de l'Organisation (DSIO). Nos missions, au sein de ce Département, sont de définir et standardiser des technologies de développement

P! : Dans quel contexte vous êtes vous lancé dans la sélection d'un outil MDD ?

M. Lipka : Sur le plan technique, nous travaillons depuis 2 ans environ sur l'évolution de nos socles techniques JEE : l'atelier de développement et de modélisation, le Framework se basant essentiellement sur des standards. Côté organisationnel, nous travaillons sur la mise en place d'un Centre de Développement pour nous aider à absorber le volume d'activité de nos développements.

P! : Quels étaient vos objectifs ?

M. Lipka : Nos objectifs principaux concernant la mise en place d'une solution MDD étaient de diminuer le temps en prise en main du Framework, d'automatiser la génération des couches techniques, de fiabiliser le développement des applications et enfin d'optimiser les abaques de développement.

P! : Quelle a été votre démarche de sélection d'une solution ?

M. Lipka : Nous avons fait un état de l'art des différentes solutions du marché et avons envoyé un appel d'offres à trois éditeurs nous paraissant représentatifs. Nous avons souhaité nous faire accompagner par un partenaire ayant une expérience réussie sur un environnement technique similaire au nôtre.

P! : Quels ont été vos critères de sélection d'une solution ?

M. Lipka : Nos critères étaient les suivants : facilité de maîtrise de la solution par nos équipes internes, pérennité, niveau de maturité des templates, activité de la communauté, sociétés partenaires maîtrisant la solution et enfin le retour sur investissement.

P! : Quels sont les principaux écueils que vous avez rencontrés ?

M. Lipka : Un marché en cours de stabilisation avec des normes et standards qui commencent à émerger Des éditeurs qui commencent à se positionner sur un outillage, mais peu de solutions packagées.

P! : Où en êtes vous dans votre démarche, quelles sont les prochaines étapes ?

M. Lipka : Nous avons fait le choix d'un éditeur et d'un partenaire nous aidant à mettre en place un prototype représentatif de nos besoins techniques. Le projet d'implémentation dépendra du résultat du prototype (gain en productivité, couverture technologique, ...)

nération systématique du code cible lors de la construction de l'application est l'une des solutions les plus directes et efficaces pour garantir que le code compilé, assemblé, déployé et qualifié est bien celui exprimé par les modèles.

Cartouches clefs en main ou à façon ?

Les éditeurs et les communautés, à la base des outils de génération dont nous parlons ici, ont développé une grande expertise dans l'art et la manière de produire du code automatiquement. Ils ne se contentent cependant pas de fournir le moteur de génération mais ils fournissent la plupart du temps des chaînes prêtes à l'usage. Elles offrent l'opportunité de démarrer très rapidement des projets de réalisation sans passer par une phase initiale de réalisation des templates et des transformations nécessaires. Elles présentent cependant deux risques significatifs. Le premier réside dans la divergence probable entre les choix d'architecture et d'implémentation retenus par l'éditeur pour les applications cibles et vos propres choix.

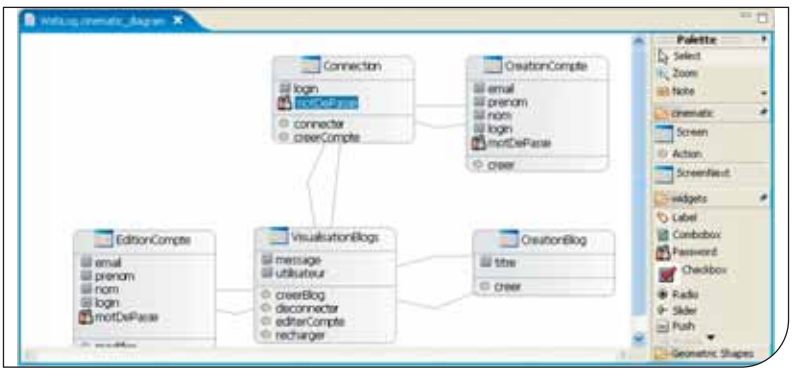
Le second risque réside dans la qualité des templates fournis. De fait, et nonobstant les compétences intrinsèques des équipes techniques des éditeurs ou des communautés, ces compétences ne sont pas nécessairement centrées sur les technologies cibles des générateurs, mais bien sur les technologies de génération ou de modélisation. En conséquence, en évitant soigneusement de verser dans le syndrome NIH (Not Invented here) il convient de se poser la question de l'exploitabilité et de l'exploitation des templates et transformations livrées avec les outils. Quelle que soit la stratégie retenue, cartouches clefs en main ou cartouches spécifiques, les facilités offertes par l'environnement de développement pour créer ou modifier ces cartouches restent primordiales pour les ajustements qui apparaîtront inévitables. Ces facilités peuvent s'appuyer sur des langages de transformations intrinsèquement flexibles (oAW) et / ou sur des outils et assistants efficaces. La capacité des outils à accepter des ajustements " locaux "aux projets, indépendamment des cartouches communes (Acceleo, oAW), représente également un réel facteur d'adaptabilité et de réactivité.

Conclusion

L'offre MDSD s'est fortement industrialisée, simplifiée et démocratisée ces quelques dernières années. Elle a entamé une structuration sur la base de normes et standards, appuyée en cela par les efforts croissants du marché et les communautés open source très actives. Ceux-ci se cristallisent tout particulièrement autour des travaux de la fondation Eclipse. La masse critique est désormais atteinte. Les approches MDSD ne sont plus réservées à une minorité très technique mais sont maintenant abordables et rentables pour tous.

■ Arnaud **Buisine**,
Directeur Technique, ProxiAD Nord, a.buisine@proxiad.com

■ Cédric **Vidal**,
Architecte Technique, expert JEE et MDSD, ProxiAD Ile de France, c.vidal@proxiad.com



Modélisation intégrée : la mettre en œuvre

Au même titre que les IDE offrent dans un cadre unifié des fonctionnalités de compilation, de débogue, de gestion de versions, de gestion de configurations…, les IME (Integrated Modeling Environment) sont les nouveaux outils CASE (Computer-Aided Software Engineering) de l'ingénierie des modèles.

Eclipse offre un support " modèle " (EMF pour Eclipse Modeling Framework) mais dont l'usage est réservé à des initiés. Le propre des IME est d'autoriser les concepteurs d'application à aller jusqu'à la méta-modélisation. Par exemple dans Eclipse, on écrit des méta-modèles à partir du méta-méta-modèle d'EMF : Ecore (cf. figure). Le méta-modèle UML lui-même est bâti dans Eclipse à partir d'Ecore. On peut aussi écrire des modèles métier à partir d'Ecore et donc finalement créer des applications orientées " utilisateurs finaux ". Dans le code qui suit, le concept métier Facture cohabite sous forme de modèle (format XML Ecore) et sous forme de code :

```
<eClassifiers xsi:type="ecore:EClass" name="Facture">…</eClassifiers>
public interface Facture extends EObject { // Etc.public class Facture
Impl extends EObjectImpl implements Facture { // Etc.
```

Un point fondamental est que les modèles comme celui de la figure 1 ne sont que des notations graphiques de dialectes XML. Parce que l'on sait à l'aide de grammaires, transformer du code en binaire, on peut aussi transformer des modèles en modèles, des modèles en texte et donc conséquemment des modèles en binaire. Ces grammaires orientées modèle ont néanmoins l'avantage d'être plus riches, plus expressives pour appréhender la complexité des besoins des utilisateurs. Les transformations de modèles s'opèrent avec des langages dédiés comme ATL (Atlas Transformation Language). Dans l'exemple qui suit, on transforme la classe Facture sans connotation technologique vers une classe Facture_avec_details_implementation teintée, elle, de propriétés technologiques :

```
module Modele_source2Modele_cible;
create mc : Modele_cible from ms : Modele_source;
rule Facture2Facture_avec_details_implementation {
  from f : Modele_source!Facture
  to fadi : Modele_cible!Facture_avec_details_implementation ( // Etc.
```

La transformation de modèles peut s'apparenter à des technologies comme XSLT qui permet de transformer un document XML en un autre mais la transformation de modèles va plus loin car il faut générer du code, traiter et interpréter la sémantique capturée dans les modèles métier, composer ces modèles métier avec des modèles techniques, ceux qui adhèrent à des plates-formes technologiques, des frameworks de persistance, des frameworks de présentation…

EMF n'est pas suffisant

L'avantage d'Eclipse / EMF est d'offrir un cadre standard et unifié pour manipuler, de manière générale, des modèles. Inconvénient : Eclipse / EMF n'est pas suffisamment intégré pour détacher le concepteur d'applications métier de tâches fastidieuses, répétitives et absconses. Si nous prenons l'environnement BLU AGE, IME bâti au dessus d'Eclipse / EMF, il prend en charge ces tâches pour laisser le concepteur d'applications focaliser sur la complexité métier. Le premier avantage est de perpétrer ce cadre normé et reconnu qu'est Eclipse / EMF. Le second bénéfice est de permettre à tous l'accès aisé et assisté à l'ingénierie des modèles. En effet, lorsque les concepteurs d'applications produisent un modèle métier dans lequel figurent les maquettes écran de l'application, les objets métier, la représentation " modèle " des enchaînements écran, des traitements à faire avant et après ces enchaînements (requêtes, calculs complexes…), des interfaces avec le système d'information existant… ils font face à une complexité telle (celle des besoins des utilisateurs finaux de l'application), qu'ils doivent organiser de façon rationnelle et efficace leur processus de développement :

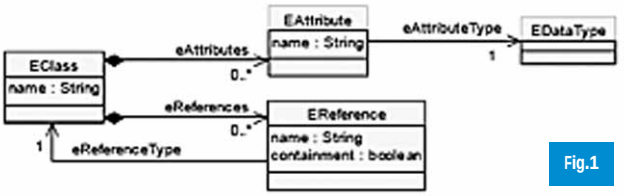


Fig.1

- travail en équipes et donc en mode partagé sur les modèles ;
- test des modèles : compilation mais surtout exécution, débogue, de manière à valider au plus tôt les besoins utilisateurs ;
- garantie de la cohérence (bidirectionnelle) entre les différents modèles, précisément : les modèles bas niveau, voire le code, doivent être une représentation fidèle, respectueuse de la sémantique des modèles les plus amont : les modèles métier ;
- gérer la traçabilité des transformations ;
- génération de code de bout en bout, synchronisation entre code et modèles. Beaucoup d'outils " modèle " ne sont en effet que des générateurs de patrons de code, patrons qu'il faut étoffer à la main par la suite, ce qui est insatisfaisant.

Le principe d'IME au cœur de BLU AGE est de fournir aux utilisateurs l'équivalent d'un IDE mais au niveau " modèle ". Toutes les fonctionnalités précitées sont fournies de manière communicante en vue d'industrialiser la production, non pas de code mais d'applications prêtes à l'emploi. BLU AGE est une usine à logiciel là où Eclipse / EMF est un atelier pour artisans expérimentés. Par exemple, BLU AGE supporte le test pas à pas de modèles par animation graphique et pose de points d'arrêt là où Eclipse / EMF offre les fonctionnalités standard des IDE.

On dispose aussi de fonctionnalités destinées aux architectes pour construire des modèles spécifiques aux technologies, fonctionnalités destinées aux experts en conformité toujours avec Ecore.

■ Franck **Barbier** - *Professeur à l'Université de PAU*
Auteur de "UML 2 et MDE - Ingénierie des modèles avec études de cas" (Dunod) Conseiller technique BLU AGE

Utiliser ou non UML ? Telle est la question !

Faut-il modéliser ? Le jeu en vaut-il vraiment la chandelle ? Voici une question qui a fait couler beaucoup d'encre dans votre magazine préféré. Je vais m'attacher ici à montrer l'intérêt de la modélisation en général et d'UML en particulier. Puis nous verrons aussi qu'UML n'est pas le Graal : il en fait parfois trop, souvent pas assez… N'hésitez donc pas à l'adapter et à le compléter par des langages spécialisés qui couvriront vos besoins spécifiques.

Pourquoi prendre le temps de modéliser ? En fait, les raisons ne manquent pas : les problèmes sont souvent plus complexes qu'on l'imagine initialement ; il faut les considérer dans leur globalité pour construire une solution efficace ... et surtout, il n'y a qu'un développeur pour se poser ce genre de question ! Pensez-vous utile de faire des plans avant de construire une maison ? Pourquoi ne pas commencer par monter quelques murs pour se faire une pièce à vivre ? Puis attaquer la construction d'une chambre pendant qu'on peindra le séjour. Note pour plus tard : faire un trou dans le mur pour accéder à la chambre ...

Caricatural ? Pas tant que ça. Beaucoup de développeurs pensent qu'ils peuvent attaquer directement par le code. Mais les modèles, ce sont les plans des applications. Exprimer clairement vos objectifs et les contraintes à respecter est indispensable pour produire une application de qualité. Sans parler d'efficacité : combien de fois avez-vous écrit une page de code pour vous rendre compte qu'il fallait tout recommencer alors qu'un schéma de synthèse vous aurait permis de détecter le problème en amont ? Et je sais de quoi je parle :-)

Tout ce qu'UML peut oser, je l'ose ?

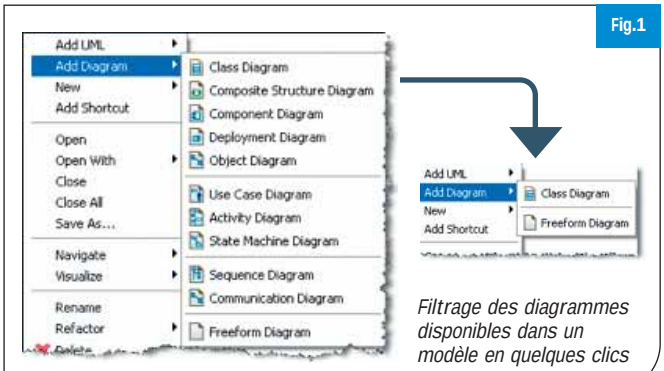
Je retiens donc UML comme base de travail. Mais j'entends déjà la voix de ses détracteurs : "c'est trop compliqué", "il faut dessiner des diagrammes qui ne servent à rien", "j'ai pas de temps à perdre", "Attention, UML inc, on va le tank, re-buff plz !!".

On a dit qu'UML était le bon langage pour modéliser un système logiciel. Mais on n'a jamais dit qu'il fallait utiliser 100% du langage ! L'atelier de modélisation et d'implémentation JEE fourni par IBM (Rational Software Architect for WebSphere Software 7.5) permet de sélectionner les éléments d'UML que vous jugez pertinents sur vos projets. Vous ne jurez que par les classes et vous vous demandez à quoi sert un diagramme de déploiement avec 3 cubes et 2 flèches qui se battent en duel : 3 clics et RSA se plie à votre volonté. Ce mécanisme fonctionne aussi bien pour les diagrammes que pour les éléments du modèle. [Fig.1]

Par ailleurs, au fil d'un projet, vous allez vous intéresser successivement aux cas d'utilisation, à l'analyse puis à la conception détaillée : le même modèle peut donc inclure tous ces éléments mais vous ne les renseignez pas au même moment. Comment adapter le rendu d'un modèle à votre activité en cours ?

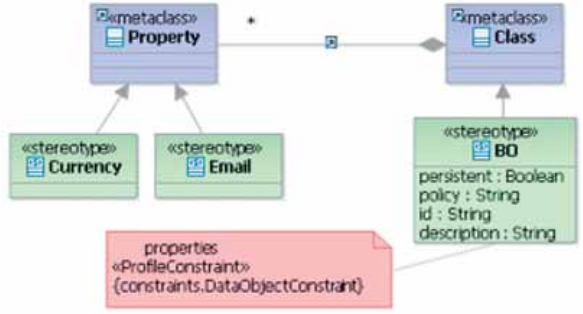
Eclipse a popularisé le principe des perspectives pour s'adapter dynamiquement à vos besoins : il existe donc une perspective Modélisation. Mais elle ne fait pas la différence entre analyse et conception. RSA 7.5 introduit la notion de "point de vue" (ou viewpoint) pour compléter les perspectives. En sélectionnant un point de vue, vous filtrez certains éléments à un instant donné et vous simplifiez l'interface graphique de votre atelier de modélisation. Simple et efficace.

Fig.1



Filtrage des diagrammes disponibles dans un modèle en quelques clics

Fig.2



Le stéréotype BO étend la notion de Classe, il inclut 4 attributs et une règle de validation.

UML contient tout juste ce qu'il faut pour l'étendre

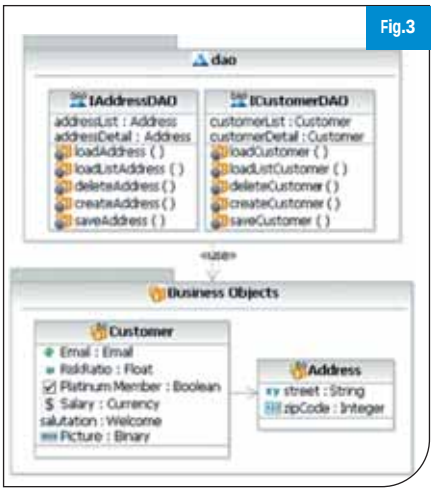
Vous avez donc sélectionné un sous-ensemble d'UML pour vos projets. Mais comment faire pour les concepts dont vous avez besoin mais que les créateurs d'UML n'ont pas jugé bon d'inclure - on se demande où ils avaient la tête !

Imaginons une application Web : le serveur héberge des traitements et des structures de données. Certains services sont dédiés à la gestion de ces données, les fameux services CRUD. Quelle est la réponse d'UML pour représenter ce scénario très classique ? La classe ! Un peu simpliste, non ?

Mais UML embarque aussi une botte secrète : les profils. Un profil inclut un ensemble de stéréotypes. Chacun enrichit un type d'entité UML (Classe, Propriété, Opération, ...) et précise sa définition avec des attributs et des règles de validation.

Reprenons notre exemple : les services CRUD sont la brique de base de la plupart des applications : ils coordonnent la création (Create), l'accès (Read), la mise à jour (Update) et la suppression des données (Delete). Comment gérer le CRUD dans un modèle ? On peut par exemple introduire la notion d'objet métier (Business Object) et préciser la définition des propriétés de ces objets. On centralise ensuite l'accès aux données avec un DAO (Data Access Object). [Fig.2]

Fig.3



On peut compléter efficacement le standard UML avec un profil.

Détail non négligeable, les profils sont faciles à mettre en œuvre; l'outil de modélisation génère même automatiquement de nouvelles palettes graphiques pour exploiter facilement vos profils. Que demande le développeur ! [Fig.3]

Il faut modéliser juste

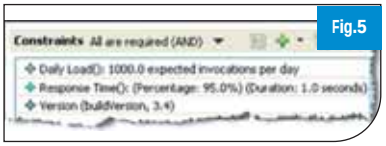
Nous avons créé des profils pour préciser la définition des éléments UML. Mais comment capturer les informations qui sortent complètement du périmètre d'UML ? En utilisant d'autres langages bien sûr.

Citons la modélisation de processus avec le standard BPMN ou la définition des interfaces graphiques avec un outil de maquettage. Autre exemple : Rational Software Architect 7.5 inclut un langage de modélisation des architectures de déploiement en complément d'UML. Vous pouvez ainsi modéliser très précisément ces informations et améliorer la communication avec les équipes opérationnelles. [Fig.4] [Fig.5]

Fig.4

IBM outille de nombreux environnements pour les architectures logiques (DB2, Tomcat, WAS, Windows, Linux ...) et les fermes de serveurs physiques.

Fig.5



Vous pouvez également définir des contraintes : charge, temps de réponse, version, mémoire vive ... L'outil vérifiera dynamiquement qu'elles sont respectées dans vos modèles.

RSA 7.5 supporte aussi la définition de nouveaux types : vous pouvez donc enrichir les palettes avec vos propres éléments dynamiques sans écrire une seule ligne de code, puis les partager avec votre équipe. [Fig.6]

En résumé, j'espère vous avoir convaincus d'utiliser UML pour modéliser vos applications. N'hésitez pas à mettre de côté les éléments qui ne vous semblent pas pertinents et complétez UML avec des profils et des langages de domaine. Vous pourrez ainsi créer des modèles synthétiques et expressifs qui vous aideront à maîtriser la complexité de vos projets informatiques. Jetez-vous à l'eau, vous ne le regretterez pas !

Deux éditions de Rational Software Architect sont disponibles en téléchargement :

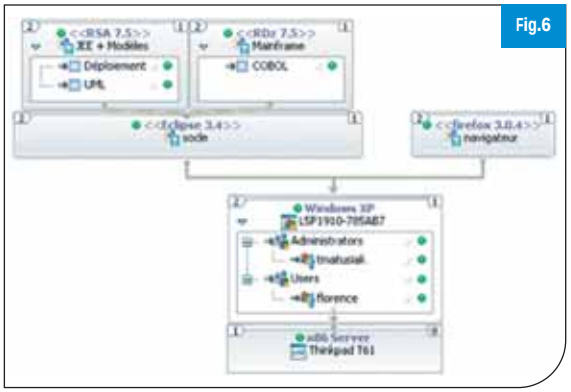
- RSA Standard Edition : http://www.ibm.com/software/awdtools/swarchitect/standard/
- RSA for WebSphere : http://www.ibm.com/software/awdtools/swarchitect/websphere/

Rational Modeling Extension for Microsoft .NET : http://www.ibm.com/software/awdtools/modeling/ext/

■ Thierry **Matusiak** travaille depuis 10 ans chez IBM dans les domaines de la modélisation, de la définition du besoin et de l'architecture d'entreprise.

Retrouvez-moi sur http://www.ibm.com/developerworks/spaces/mdd

Fig.6



Prenons un exemple quelconque : ma machine ! J'écris cet article sur un Thinkpad T61 sur lequel sont déclarés 2 utilisateurs Windows XP. Sur le même socle Eclipse 3.4, j'ai installé RSA 7.5 et RDz 7.5, l'atelier IBM pour les développements Mainframe.

Panorama des outils

Sur le marché de la modélisation il existe de nombreux outils pour l'UML et toutes les mouvances Model Driven. Nous vous proposons une sélection de solutions de modélisation commerciales et open source. L'offre est particulièrement importante aussi bien dans le monde open source que commercial. Les modeleurs UML sont les plus nombreux, même si aujourd'hui beaucoup d'outils mêlent UML, approche MD, génération de code, etc. Bref, le choix ne manque pas.

A vous de tester et de trouver l'outil idéal correspondant au mieux à vos besoins, vos contraintes projets. Si vos équipes ne sont pas **" Model Ready ",** une formation sera à envisager. Il s'agit d'un marché mouvant et les solutions naissent et disparaissent.

C'est notamment le cas d'OptimalJ, un des pionniers du MDA, qui a été arrêté par Compuware courant 2008. Mieux vaut opter pour des versions récentes.

Les solutions et outils UML

Nom de la solution	Licence	Commentaire
PowerAMC (Sybase)	commerciale	Outil historique, PowerAMC permet de modéliser et de désigner des processus d'entreprise, les données. Très complet et reconnu par le marché, il supporte UML et Merise. PowerAMC 15 est disponible depuis l'automne.
MagicDraw (No Magic Inc.)	Commerciale	Idéal pour le travail en équipe, les process métiers, le design objet, les systèmes, les données, autorise le round trip et supporte de nombreux langages. Très complet. Version 16 sortie en automne._
Win'Design (Cecima)	commerciale	Doté de 4 modules (database, object, business process et user interface), Win'Design est un modeleur multi usage (bpm, données, IHM, Merise, UML). Version 9 prévue courant janvier 2009.
UModel (Altova)	commerciale	Modeleur UML2 générant du Java, C#, VB.Net, avec reverse ingeniering, support modèle XML, XMI 2.1, intégration Eclipse, Visual Studio.
JViews (Ilog)	Commerciale	L'éditeur français intègre dans Jviews des éditeurs UML / EMF sous Eclipse pour modéliser les activités de l'entreprise plus facilement.
Visual Paradigm for UML (Visual Paradigm)	gratuite	Dans la version communautaire, l'outil propose les diagrammes de base, les flux de données, le process métiers et quelques possibilités de styles et de formatage des diagrammes.
Poseidon (gentleware)	Open source / commerciale	Modeleur UML comportant 9 diagrammes, la navigation, la génération de code Java, l'export des modèles. Installation simplifiée.
ArgoUML	Open source	Outil de modélisation permet de démarrer en douceur dans UML même si le projet est encore en développement !
Enterprise Architect (Sparx)	Commerciale	Modeleur UML 2.1 complet avec génération de code (java, .Net, Delphi, etc.). Supporte le modelage en équipe, XMI et dispose d'une API pour étendre l'outil.
Papyrus 4 UML (CEA)	Open source	Basé sur Eclipse, Papyrus est un modeleur UML 2 respectant DI2, extensible et acceptant les profils UML.
Umbrello	Open source	Modeleur pour KDE. Supporte différents langages dont PHP5, C#, Java, C++.
StarUML	Open source	Plate-forme UML et MDA. Extensible et fonctionnant sous Windows. Disponibilité de Template Designer.
JDeveloper (Oracle)	gratuite	Environnement d'Oracle. Propose un puissant modeleur UML réécrit dans en 11g. Supporte Java. Idéal pour concevoir des modèles de données.

Outils Model-Driven

Offre	Licence	Editeur	Type	Meta meta modèle	Transformation modèle à texte	Transformation modèle à modèle	Intégration IDE(1)	Modeleur
Acceleo	Open source	Obeo	Pure player MDSD	Ecore (2)	Propriétaire	non	Eclipse	ouvert
Acceleo Pro	Commerciale	Obeo	Pure player MDSD	Ecore	Propriétaire	Propriétaire (ATL annoncé)	Eclipse	ouvert
AndroMDA 3.x	Open source	-	Pure player MDSD	MOF (3)	Velocity	non	ouvert	MagicDraw Poseidon
ArcStyler	Commerciale	Interactive Objects	Pure player MDSD	MOF	Propriétaire	QVT	IBM RSM, MagicDraw	ouvert
Blu Age	Commerciale	Netfective Technology	Pure player MDSD	MOF & Ecore	JET	QVT (ATL annoncé)	Eclipse	ouvert
Eclipse M2T & M2M	Open source	Eclipse	Pure player MDSD	Ecore	JET	ATL, QVT	Eclipse	ouvert
MDWorkbench	Commerciale	Sodius	Pure player MDSD	Ecore	Propriétaire	Propriétaire & ATL	Eclipse	ouvert
Mia-Studio	Commerciale	Mia Software	Pure player MDSD	Ecore & MOF	Propriétaire	Propriétaire	Eclipse	ouvert
oAW	Open source	-	Pure player MDSD	Ecore	Xpand	Propriétaire (Xtend)	Eclipse	ouvert
RSM et RSA	Commerciale	IBM	Modeleur et IDE	Ecore	JET	Propriétaire	Eclipse/RSA	ouvert
Together Visual Studio	Commerciale	Borland	Modeleur et IDE	Ecore	Xpand , JET	QVT	Eclipse	ouvert
DSL Tools	Commerciale	Microsoft	Modeleur et IDE	Microsoft DSL	Propriétaire	non	Visual Studio	Visual Studio
Objecteering	Commerciale	Objecteering Software	Modeleur	n.c.	Propriétaire	Propriétaire	Eclipse Visual Studio	Objecteering

Ce tableau a été réalisé par Arnaud Buisine (ProxiAD)

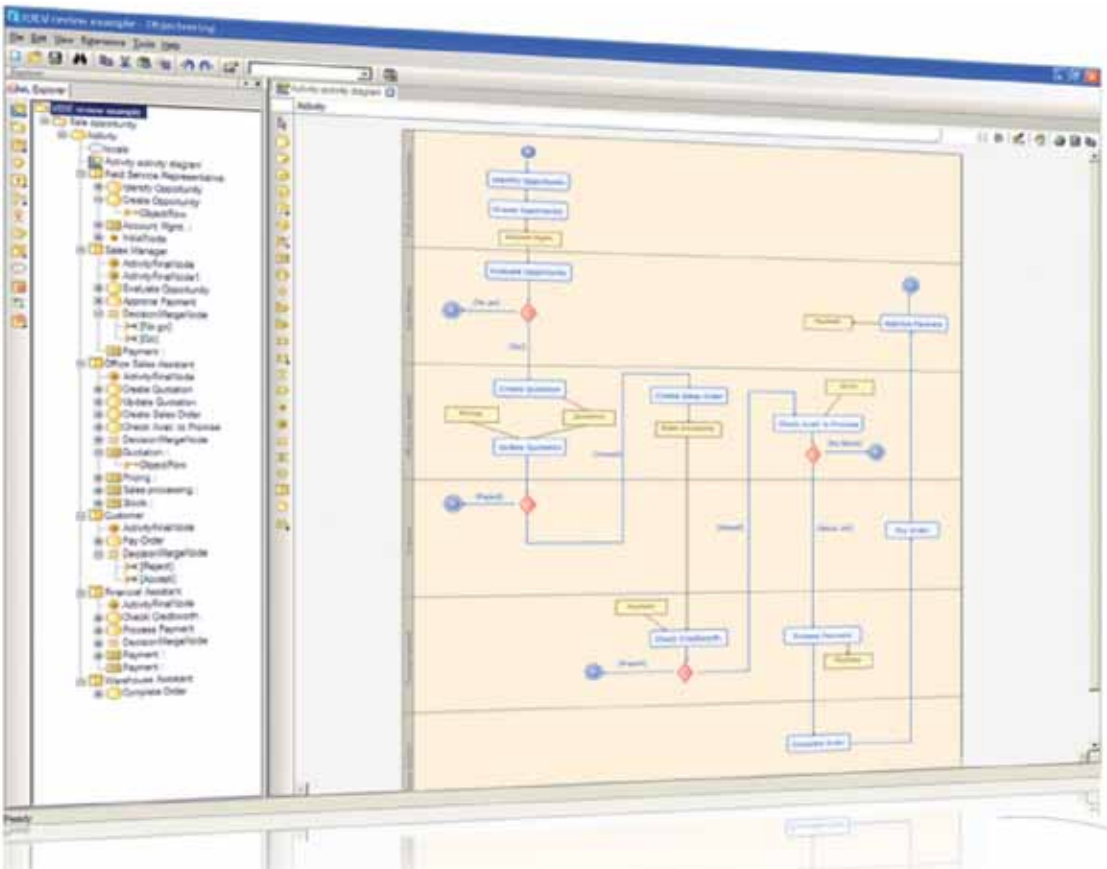
(1) IDE : Integrated Development Environment – Environnement de Développement Intégré
(2) Ecore : MMM du Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/?project=emf
(3) MOF : Meta-Object Facility, MMM normalisé par l'OMG, http://www.omg.org/spec/MOF/2.0/

Quelques autres outils

Nom de la solution	Licence	Commentaire
CodeFluent (Softfluent)	commerciale	Génération des applications .Net par modèles qui sont des schémas structurés XML, avec choix des briques techniques. Un outil particulièrement puissant !
Leonardi	gratuite	Générer automatiquement les IHM de vos applications grâce à l'approche modèle MMI.
(W4 Lyria)		Supporte de nombreuses interfaces Java et Web.
Windev (PCSoft)	commerciale	L'atelier clé en main de PC Soft intègre des modules de modélisation Merise et UML pour faciliter le développement des applications !
ER/Studio (Embarcadero)	Commerciale	Environnement complet de modélisation de données. Supporte l'approche MD, le cycle de vie complet des données ou encore un module de qualité du design des bases.

Your projects deserve a tool*

Objecteering Modeler



Objecteering 6.1

guide vos développements par les modèles sur toute la portée du système

Objecteering 6.1 est une solution de modélisation complète et ouverte supportant les standards OMG de modélisation UML 2.1, BPMN, et SysML, la modélisation de l'Architecture d'Entreprise et la modélisation d'une architecture SOA.
Il intègre le support de l'analyse des besoins et de la définition du dictionnaire, assurant ainsi une traçabilité complète sur tout le cycle de vie.
La technologie MDA associée à une ouverture de l'outillage en Java permet de guider le développement à chaque contexte utilisateur, chaque infrastructure technique cible. Ses générateurs sur étagère automatisent le développement d'applications Java, J2EE, .Net C#, C++, SQL...

Objecteering Software, éditeur de l'atelier Objecteering 6, est le spécialiste français UML/MDA pour le développement d'applications guidé par le modèle. Son offre modulaire couvre le cycle de vie de la gestion des exigences jusqu'au déploiement d'application.
Pour plus d'information sur Objecteering 6.1, pour télécharger Objecteering **Free Edition** ou Objecteering **Enterprise Edition**, rendez-vous sur : www.objecteering.com

Architecture d'Entreprise
UML2 BPMN
C# MDA SysML
Java SQL
J2EE SOA
C++

Quinte&sens • 4721

* Vos projets méritent un outil

www.objecteering.com - Tél. : 01 30 12 16 60 - sales@objecteering.com

Objecteering SOFTWARE

The model-driven development company

Dopez votre site web à la vidéo

Sur le CD virtuel

Difficile de passer à côté de la vidéo sur le web. Elle est partout. Des sites comme Dailymotion ou YouTube ont permis cette explosion auprès du grand public. Il existe plusieurs formats pour la diffuser et de nombreux outils de montage, d'encodage. Mais seules trois principales plates-formes existent : Adobe, Apple et Microsoft. Que signifie pour un webmaster, un développeur, la vidéo sur le web ? Quel intérêt de créer sa web TV ? Comme nous le verrons dans ce mini dossier, le monde de la vidéo est un univers étonnant. Nous aborderons Silverlight et surtout les API YouTube !

Embarquer une vidéo dans un site web n'est guère difficile. Il suffit d'intégrer les quelques lignes de code comme YouTube en fournit. Mais dès que l'on passe à l'étape au-dessus, créer son propre flux vidéo, là, l'affaire se corse quelque peu. Faut-il créer son player ? Quelle plate-forme et outils utiliser ? Vidéo streaming, ou à chargement progressif ? Etc. Les questions ne manquent pas et si le développeur n'est pas habitué, il risque d'y perdre... son frame. Mais coder les API YouTube ne va pas toujours de soi. *" Utiliser YouTube ou Dailymotion est intéressant notamment par l'absence de flux réseau à gérer (soi-même) "* précise **Arnaud Barbier,** Directeur Associé de ViewOnTV. *" Par contre, pour son image ce n'est pas forcément une bonne idée ! Pour du marketing viral, pourquoi pas. Pour de la vidéo d'entreprise, l'audience sera faible, ce n'est pas l'Eldorado de la visibilité. "*

La chaîne vidéo

En gros, il y a trois points à considérer : la source, l'encodage et la diffusion. La source est votre vidéo. Celle-ci peut prendre plusieurs formes : flux en direct (dit " live ") pour retransmettre par exemple une conférence, séquence vidéo provenant d'un caméscope (dv, broadcast, webcam, etc.) et enfin une vidéo préexistante dans un format donné (flv, mov, wmv, mpeg, etc.). *" Plus la source sera de bonne qualité, plus le résultat (après encodage) sera bon "* précise de **Xavier Pouyat** (Microsoft). Par contre encoder une vidéo déjà encodée provoquera une perte de qualité à la sortie.

L'encodage dépend du client utilisé pour diffuser (Silverlight, Flash par exemple). Car selon le client visé il faudra utiliser le bon format vidéo en sortie d'encodage. Par exemple, pour du Flash ou pour une large diffusion, préférez le format flv, avec Silverlight, se sera du WMV en priorité... Ensuite, il faut trouver les bons outils. Parmi les plus connus, du moins dans le monde Flash, on a Flash Media Live Encoder. En Silverlight ce sera Expression Encoder, ou encore QuickTime Pro pour la partie Apple. Vous pouvez aussi utiliser des outils open source tels que VLC ou encore MPEG4IP, ffmpeg. Restera à bien configurer le flux vidéo selon la bande passante réseau (il faut penser aux connexions à débit limité et pas uniquement haut débit ADSL 8 Mo ou plus). Le temps d'encodage peut être très long selon la durée de la vidéo, le format, l'outil, les paramètres !

Une fois la vidéo encodée, il faut maintenant la diffuser. Là encore plusieurs solutions. Vous pouvez la déployer directement sur le site et placer un viewer sur la page html. Pour une diffusion plus large et affinée, il faudra passer par un serveur de streaming. La diffusion peut se faire en deux modes : un téléchargement progressif ou en streaming. Le streaming nécessite un serveur de streaming. Dès que vous avez un nombre important de vidéos, mieux vaut passer par un serveur vidéo. De nombreux hébergeurs proposent le support vidéo sur leurs serveurs (ex. : Ikoula). Dans le cadre de Silverlight vous pouvez passer par un service de type Silverlight Streaming. Côté Flash, vous pouvez utiliser un serveur open source tel que Red5, chez Apple, il existe Darwin Streaming Server.

Par contre, dès que vous passez à la diffusion en direct, là, l'infrastructure sera plus lourde, notamment au niveau capture (via une carte d'acquisition professionnelle), avec une puissante machine d'encodage. La diffusion se fera via un serveur de streaming, les principaux éditeurs du domaine en proposent, avec quelques solutions open source. Le coût d'une telle infrastructure est très variable, de quelques centaines d'euros à plusieurs milliers !

François **Tonic**

Silverlight Streaming et Silverlight 2 : tout par la vidéo

Silverlight Streaming est un service de Microsoft qui permet aux webmasters et développeurs de diffuser leurs vidéos ainsi que leurs applications Silverlight 2 sur Internet afin de les rendre accessibles à tous.

Silverlight streaming propose un modèle gratuit jusqu'à 10Go de stockage de streaming. Ce service nécessite un compte LiveID. A terme, il sera possible d'outrepasser ces limitations de stockage via des options payantes.

Pourquoi profiter du service Silverlight Streaming ?

Il s'agit d'une solution gratuite d'hébergement d'application Silverlight et de vidéo. Dans un contexte de consommation, le service Silverlight Streaming s'adapte à la demande des utilisateurs pour fournir un service performant.

Suivant le lieu où vous vous trouvez Microsoft sélectionne le serveur adéquat. C'est ce qu'on appelle le système CDN (Content Delivery Network) qui permet une mise en cache similaire aux services Akamai. Ainsi, fini les temps d'attente interminables !

Comment profiter de ce service

Il suffit de vous inscrire sur le site http://streaming.live.com/ avec votre compte LiveID. Une fois inscrit sur le service, un identifiant ainsi qu'une clé vous sont attribués.

Ces identifiants vont vous permettre d'accéder au service via l'interface web proposée sur le site ou bien via une API.[Fig.1]

Héberger votre application Silverlight ou votre vidéo sur Silverlight Streaming.

Comme énoncé ci-dessus, Silverlight Streaming donne la possibilité d'héberger n'importe quelle application Silverlight ou n'importe quelle vidéo. Cependant, vous allez devoir faire quelques adaptations avant d'exposer votre application sur la plate-forme Live.

1. Au sein de votre application il va falloir utiliser un fichier " manifest ". Ce fichier va permettre à la plate-forme Live de générer la page recevant votre application correctement. Voici un exemple du fichier " manifest.xml " :

```
<SilverlightApp>
    <source>VotreFichier.xap</source>
    <version>2.0</version>
    <width>400</width>
    <height>200</height>
</SilverlightApp>
```

[Fig.2]

2. Vous allez aussi devoir vous assurer que tous les formats de fichier présents dans votre application sont autorisés par le service Silverlight Streaming. Le tableau de la page suivante vous donne la liste de ces formats.

Fig.1

Fig.2

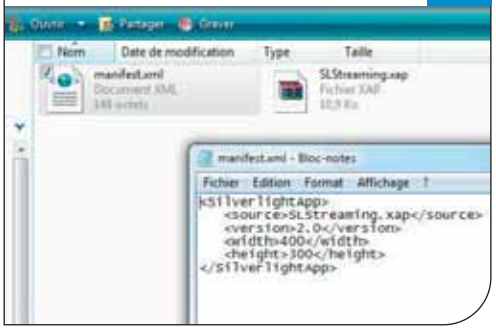


Fig.3

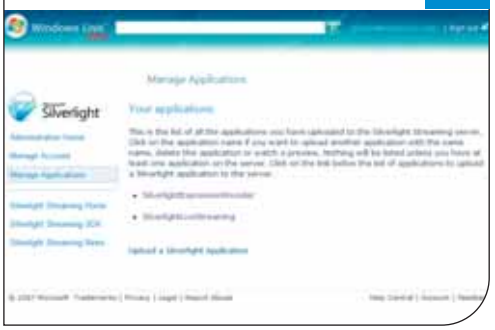


Fig.4

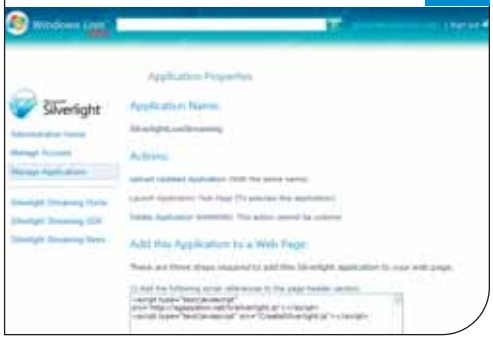


Fig.5

Fig.6

Extension	MIME Type
ttf	application / octet-stream
dttf	application / octet-stream
js	application / x-javascript
dll	application / x-msdownload
xaml	application / xaml+xml
xap	application / x-silverlight-app
xml	application / xml
zip	application / x-zip-compressed
wma	audio / x-ms-wma
wmv	audio / x-ms-wmv
mp3	audio / mpeg
gif	image / gif
jpeg	image / jpeg
png	image / png
txt	text / plain
bin	text / plain
sdx	text / plain

A noter que si l'upload du fichier dure plus de 15 minutes, une erreur survient. Pour information, la MSDN nous précise qu'une vidéo de 22MO mettra 10 minutes à être uploadée avec une connexion de 300 Kbps.

Rappel : Comme vous pouvez le remarquer dans le tableau ci-dessus, les .HTML ne sont pas acceptés. C'est le service Silverlight Streaming qui va lui-même se charger de générer le fichier qui hébergera votre application.

3 - Une fois ces points traités vous pouvez ZIPer votre application et l'uploader via l'interface web si le ZIP pèse moins de 30MO. [Fig.3] Votre application est en ligne ! Dès lors une page vous explique comment insérer votre application au sein d'une page web via le service Live Streaming. Cependant, vous remarquerez que la solution proposée nécessite d'avoir accès à la partie <head> de votre page HTML. [Fig.4] Or, les bloggeurs le savent bien cette partie n'est pas souvent accessible lorsqu'on n'est pas propriétaire de son web / blog. Je vous propose donc en attendant une solution plus adéquate de passer par une IFRAME.


Fig.7

Fig.8

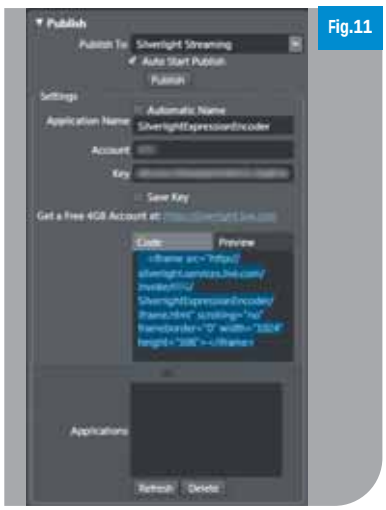


Fig.11


Fig.9


Fig.10

```
<iframe  src="http://silverlight.services.live.com/invoke/<Account LiveStreaming>/<Nom de l'application>/iframe.html" scrolling="no" frameborder="0" width="400" height="200"></iframe>
```

Pour vérifier le bon fonctionnement de votre application le service Silverlight Streaming génère automatiquement une page de test appelée via le lien " Test application". [Fig.5] Notez que si vous oubliez le fichier *manifest* au sein de votre application le service vous proposera d'en créer un automatiquement. [Fig.6]

4 - Pour une vidéo la manipulation est beaucoup plus simple, il suffit d'uploader votre fichier WMV dans la rubrique adéquate : " Manage Videos " et une URL statique vous est fournie pour accéder au fichier vidéo.

Héberger votre player Silverlight sur la plate-forme Streaming Live via Expression Encoder

Par défaut, le logiciel Expression Encoder ne permet pas de publier une application Silverlight directement sur le service Live Streaming. Pour pallier ce manque il existe un plug-in mis à disposition par Microsoft permettant d'ajouter cette fonctionnalité au logiciel. Ce plug-in, Silverlight Streaming Publishing Plug-In for Expression Encoder, est téléchargeable à l'adresse suivante :
http://www.microsoft.com/downloads/details.aspx?familyid=0708E7D7-9BA1-448E-9C82-3D71E8979A1B&displaylang=en

Il s'agit d'un fichier MSI qui va ajouter une DLL appelée " SLSPublishPlugin.dll " dans le répertoire " C:\Program Files\Microsoft Expression\Encoder 2\Plugins " de votre machine. Cette manipulation va donc vous permettre de retrouver des nouveaux menus au sein de l'interface du logiciel Expression Encoder, notamment en ce qui concerne la partie de publication.[Fig.7]. Une nouvelle partie intitulée " Publish " apparaît dans l'onglet " Output " et permet donc de renseigner la façon dont le document sera publié. [Fig.8] Le plug-in installé permet ainsi de publier vers Silverlight Stream via une API. Il ne reste plus qu'à renseigner vos identifiants Windows Live ID et d'encoder la vidéo [Fig.9]. Surtout, n'oubliez pas de renseigner un template pour votre player ! [Fig.10] Une fois l'encodage et donc la publication terminée Expression Encoder génère le HTML à copier-coller dans vote site web et vous permet d'avoir un mini aperçu de votre application [Fig.11].

En résumé :

- Installation du plug-in "Silverlight Streaming Publishing Plug-In for Expression Encoder 2"
- Création d'un projet Expression Encoder classique, insertion de la vidéo, mise en place des markers, etc.
- Renseignement des identifiants Live Streaming
- Choix d'un template pour le player Silverlight
- Encodage et récupération du code HTML permettant d'insérer l'application dans votre page web.

Quelques liens à retenir

- Silverlight Streaming http://streaming.live.com/
- Silverlight Streaming pour les développeurs http://dev.live.com/silverlight/
- Microsoft Silverlight Streaming SDK
http://msdn2.microsoft.com/en-us/library/bb851621.aspx
- Silverlight Streaming REST API
http://msdn2.microsoft.com/en-us/library/bb851616.aspx

Guillaume **André** - *Développeur RIA*

Développer avec les API YouTube !

Nous connaissons tous YouTube et la possibilité d'ajouter rapidement des vidéos provenant de YouTube dans son blog, son site web. Mais finalement peu de développeurs connaissent réellement la richesse de l'API YouTube, c'est ce que nous allons vous montrer dans le présent article !

Une api composée de 4 blocs

Il n'existe pas une API Youtube mais des API. Pour être plus précis, il en existe quatre :

- La **data API** vous permet d'accéder aux fonctionnalités que vous retrouvez sur le site de Youtube : recherche, upload de vidéos, authentification d'un utilisateur, modification d'une playlist, récupération de feeds, etc.
- **L'API du player** Youtube vous permet de modifier le look du player. Là encore, Youtube scinde cette API en deux :
  - Embed Player que vous connaissez tous. Ce player peut être personnalisé en modifiant différents paramètres (cf.http://code.google.com/apis/youtube/player_parameters.html).
  - Chromeless player qui est une vidéo Youtube dénuée de tout contrôle. Vous avez alors la possibilité d'utiliser une API accessible en JavaScript ou ActionScript pour contrôler la vidéo (pause, play, stop etc.).
- **Custom player** est une aide afin de personnaliser votre player.
- **Les Widgets** vous permettent de prendre certaines parties de Youtube pour les mettre sur votre site. A titre d'exemple, vous pouvez installer un Widget de recherche sur votre site web (cf. http://www.google.com/uds/solutions/videosearch/index.html).

Les deux dernières API n'étant pas directement destinées aux développeurs, nous allons nous concentrer sur l'utilisation des deux premières, via un exemple très simple d'intégration de Youtube dans un site Web.

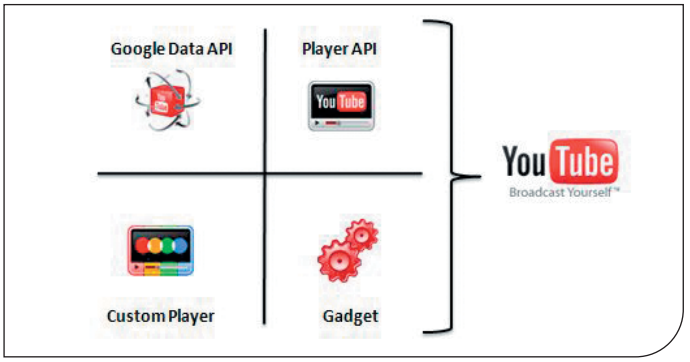
Avant de commencer

Google fournit plusieurs exemples de codes dans différents langages de programmation. Ainsi vous retrouverez des exemples d'utilisations en Java, .NET, PHP et Python. Les exemples de notre article seront en PHP.

- Si vous utilisez Java, vous devrez avoir la version 1.5. Vous pouvez trouver le package pour pouvoir utiliser l'API ici : http://code.google.com/p/gdata-java-client/downloads/list.
- Si vous préférez .NET 2.0 ou supérieur, vous pouvez télécharger la Google Library sur l'url suivante : http://code.google.com/p/google-gdata/downloads/list. Après avoir ajouté la DLL à votre projet, vous pourrez utiliser l'API Youtube en ajoutant les " using " suivants :

```
using Google.GData.Client;
using Google.GData.Extensions;
using Google.GData.YouTube;
using Google.GData.Extensions.MediaRss;
```

- Avec Python 2.2+ vous avez besoin de télécharger la Google API Library pour Python disponible à cette adresse : http://code.google.com/p/gdata-python-client/. De plus, vous aurez besoin des éléments suivants de Python : Element Tree, httplib et urllib. Plus d'information : http://code.google.com/p/gdata-python-client/wiki/DependencyModules
- Avec PHP, vous devez posséder la version 5.1.4 ou supérieure en plus du pack ZendGdata-1.7.1 que vous trouverez à cette url : http://framework.zend.com/download/gdata.



Initialisation de l'API Youtube

En premier lieu, avant de commencer la logique de votre application vous devez bien instancier tout ce qu'il faut. Le chemin d'accès à la librairie GData doit être déclaré dans le include_path (cf. http://code.google.com/support/bin/answer.py?answer=76585#appendix pour plus d'information). Ensuite, vous pouvez l'inclure dans votre programme :

```
//Chemin d'accès de la librairie Zend :
$zend = "Zend";
require_once $zend.'/Loader.php';
```

Ceci fait, il vous suffit d'instancier l'objet **Zend_Gdata_YouTube** pour pouvoir utiliser l'API Youtube :

```
Zend_Loader::loadClass('Zend_Gdata_YouTube');
$yt = new Zend_Gdata_YouTube();
```

Faire une recherche dans Youtube

Pour effectuer une recherche dans Youtube vous devez créer une requête à l'aide de newVideoQuery. Un peu comme on a l'habitude de le faire pour SQL :

```
// On crée la requête à l'aide de l'API Youtube
$query = $Youtube->newVideoQuery();
$query->setOrderBy($order); // dans l'ordre demandé
$query->setMaxResults($number_page); //nombre par page
$query->setStartIndex($index); //index de début pour les pages
$query->setVideoQuery($keywords);
$videoFeed = $Youtube->getVideoFeed($query);
//On affiche le résultat
echo "<h2>Résultats de la recherche pour ".$keywords."</h2>";
printVideoFeed($videoFeed);
```

La fonction *printVideoFeed* permet d'afficher le feed de vidéo que la requête nous renvoie. Sachez qu'il y a plusieurs images qui peuvent servir de *Thumbnail* et qu'il faut en choisir un. Dans l'exemple de code suivant, nous choisissons un *Thumbnail* au hasard :

```
function printVideoFeed($videoFeed)
{
 foreach ($videoFeed as $videoEntry)
 {
   printVideoEntry($videoEntry);
   echo "<hr />";
 }
}
/*Affiche une vidéo à partir d'une entrée*/
function printVideoEntry($videoEntry)
{
 //Récupérer les images d'aperçu
 $videoThumbs = $videoEntry->getVideoThumbnails();
 $thumbs = array();
foreach($videoThumbs as $videoThumb)
 {
   if($videoThumb['height'] < 100) //uniquement les petites images
     $thumbs[] = $videoThumb['url'];
 }

 //Choisir une image au hasard
 $nb_thumbs = count($img_thumbs);
 $choice = rand(0, $nb_thumbs-1);
 $img_thumb = $thumbs[$choice];
 echo"<div class='resultat'>
   <div class='image'><img src='".$img_thumb."' alt='thumb' /></div>
   <div class='description'><h3>".$videoEntry->getVideoTitle()."</h3>
   <p>".$videoEntry->getVideoDescription()."</p>
   <p><strong><a href='?video_id=".$videoEntry->getVideoId()."'>Voir
la     vidéo</a></strong></p></div></div>";
}
```

Cette partie du code correspond à l'utilisation de l'API Data de Youtube. Avec cette API, vous pouvez également uploader des vidéos, ajouter des informations sur celle-ci ou encore la modifier. Vous pouvez également manipuler les informations des utilisateurs (playlist, profile etc.). Bref L'API Data de Youtube est vraiment très complète et facile à prendre en main. Pour avoir plus d'information sur cette partie, je vous invite à consulter le lien suivant : http://code.google.com/intl/fr-FR/apis/youtube/1.0/developers_guide_php.html.

Personnalisez votre player grâce à JavaScript

Après avoir effectué une recherche sur Youtube, il semble logique de vouloir afficher cette vidéo afin de pouvoir la visualiser. A l'aide de l'API du Player, vous pouvez créer un player tout à fait personnalisé grâce à des appels en JavaScript ou ActionScript.

Pour afficher le player Youtube dans votre page, vous devez posséder SWFObject (cf. http://code.google.com/p/swfobject/). C'est cet objet dont nous avons besoin pour afficher une vidéo Youtube :

```
<script type="text/javascript" src="swfobject.js"></script>
```

Il faut ensuite instancier cet objet :

```
<script type="text/javascript">
    var params = { allowScriptAccess: "always" };
    var atts = { id: "myytplayer" };
   //Lecteur Chromeless
    swfobject.embedSWF("http://www.youtube.com/apiplayer?enab
lejsapi=1&playerapiid=ytplayer",
    "ytapiplayer", "425", "356", "8", null, null, params, atts);
</script>
```

Une fois l'objet instancié, un événement est déclenché. Vous pouvez traiter cet événement en créant la fonction **onYouTubePlayerReady :**

```
//Quand le player sera chargé...
function onYouTubePlayerReady(playerId)
{
 //Sauver le lecteur dans une variable
   ytplayer = document.getElementById("myytplayer");
 //Charger la vidéo
   ytplayer.cueVideoById("'.$video_id.'"); // On prend le video_id de l'url.
   setInterval(updateytplayerInfo, 250); //fréquence de mise
 à jour des infos du lecteur
   updateytplayerInfo();
   ytplayer.addEventListener("onStateChange", "onytplayerState
Change");
}
```

On ajoute un événement pour connaître l'état de la vidéo :

```
//Met à jour l\état du lecteur quand il change
function onytplayerStateChange(newState) {
 document.getElementById("playerstate").innerHTML = newState;
}
```

Nous avons également besoin d'une fonction pour récupérer les informations sur la vidéo comme le volume ou encore la durée :

```
//Met à jour les champs infos de la page
function updateytplayerInfo() {
 document.getElementById("duration").innerHTML = getDuration();
 document.getElementById("currenttime").innerHTML = getCurrentTime();
 document.getElementById("bytesloaded").innerHTML = getBytesLoaded();
 document.getElementById("bytestotal").innerHTML = getBytesTotal();
 document.getElementById("startbytes").innerHTML = getStartBytes();
 document.getElementById("volume").innerHTML = getVolume();
}
```

Il faut ensuite traiter les événements sur les boutons play, pause etc. Pour cela, rien de plus simple :

```
function play() {
   if (ytplayer) {
               ytplayer.playVideo();
   }
}
function pause() {
    if (ytplayer) {
 ytplayer.pauseVideo();
    }
}
```

Vous pouvez contrôler bien plus d'événements comme le bouton stop, next etc. Et rien ne vous empêche d'intégrer YouTube à vos applications Google Maps ou encore avec Google Gears ! Une autre dimension des applications web s'ouvre à vous ! Pour plus d'information : http://code.google.com/apis/youtube/overview.html

■ Loïc **Bar** et Caroline **Lomba** - *Wipus*

DÉVELOPPEZ VOTRE SAVOIR-FAIRE

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 !

ATTENTION ! Tarifs en hausse le mois prochain !

11 numéros par an : **45 €***

soit **3 Numéros GRATUITS**

**Tarif France métropolitaine*

Choisissez votre formule

- **Abonnement 1 an au magazine : 45 €**
  (au lieu de 65,45 € tarif au numéro) *Tarif France métropolitaine*
- **Abonnement Intégral : 1 an au magazine + archives sur Internet et PDF : 57 €** *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 1€ par mois !

Cette option est réservée aux abonnés pour 1 an au magazine, quel que soit le type d'abonnement (Éco, Numérique, Etudiant). Le prix de leur abonnement normal est majoré de 12 € (prix de lancement, 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 aussi vous abonner en ligne et trouver tous les tarifs www.programmez.com

PROGRAMMEZ

- ☐ **Abonnement 1 an au magazine : 45 €** (au lieu de 65,45 € tarif au numéro) *Tarif France métropolitaine*
- ☐ **Abonnement Intégral : 1 an au magazine + archives sur Internet et PDF : 57 €** *Tarif France métropolitaine*
- ☐ **Abonnement Etudiant : 1 an au magazine : 39 €** (au lieu de 65,45 € tarif au numéro) *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

PROgrammez ! Le magazine du développement

Offre limitée, valable jusqu'au 31 janvier 2009

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 115

Faites de la vidéo en live sur votre site web !

Insérer une communication streamée en direct dans un site web n'est normalement pas une tâche facile. Il faut soit installer un logiciel spécifique, soit des connaissances en Flash, en ActionScript et l'utilisation d'un serveur de streaming. Maintenant, le service Apideo vous permet de faire du streaming en quelques lignes de Javascript. Dans cet article, nous allons étudier comment insérer une communication vidéo entre 2 utilisateurs dans votre site web à l'aide de l'API Apideo.

La solution est une API destinée à faire de la Vidéo sur le web (le nom vient de API-viDEO). Pour utiliser le service, il faut s'inscrire sur le site web (http://www.apideo.com/signup), pour récupérer une clef. Cette clef sera utilisée dans le code Javascript pour accéder au service. Apideo fonctionnant uniquement en Javascript, on pourra l'intégrer dans n'importe quel site web, qu'il soit développé en PHP, Java, .NET, Ruby... ou même en HTML statique. Les clefs sont gratuites pour toute utilisation non commerciale !

Un peu de théorie

Téléchargez le package Apideo sur le site web (http://www.apideo.com/download). Le package est composé de deux fichiers Javascript et de 3 fichiers flash, qui doivent être copiés dans votre site web. Pour cet exemple, imaginons que votre site web soit disponible à l'adresse http://monsiteweb.com.

La première étape va consister à copier les fichiers du package sur le site web, par exemple dans un répertoire " apideo ". Nous supposerons donc que le fichier apideo.js est accessible via internet à l'adresse http://monsiteweb.com/[chemin vers apideo]/apideo/apideo.js.

Ainsi, lorsqu'un utilisateur se connecte à votre site web, il commencera par télécharger les fichiers Apideo présents sur votre site. L'API se connectera ensuite automatiquement aux serveurs de Apideo pour diffuser le contenu (voir Schéma 1). Le flux vidéo ne transite pas par votre serveur, ce qui permet de conserver la bande passante de celui-ci.

La structure objet

L'API est composée de 5 objets principaux. L'objet Apideo est la racine de l'API. C'est cet objet que l'on utilise pour se connecter aux serveurs Apideo, en passant en paramètre la clef. La procédure de connexion retourne une " ApideoConnection ", c'est-à-dire un objet permettant de retourner des " Room ". Une " Room " est l'élément de base. C'est un espace virtuel sur lequel plusieurs utilisateurs peuvent se connecter, échanger des flux vidéos ou des évènements Javascript. Dans notre exemple, nous allons donc connecter les deux utilisateurs à une room. Enfin, l'API fournit 2 objets créés par la Room : ApideoCamera, qui représente un flux filmé par la webcam de votre ordinateur, et ApideoViewer, qui représente un flux chargé par votre ordinateur.

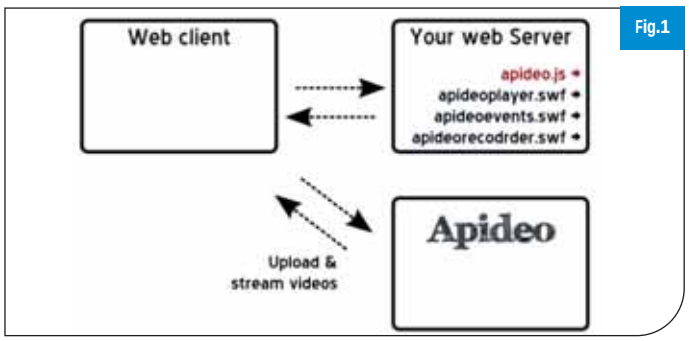


Fig.1

Un peu de pratique

Pour simplifier cet exemple, nous allons développer 2 pages HTML symétriques. La page " alice.html " est utilisée par Alice. Elle envoie le flux vidéo nommé " aliceStream " et reçoit le flux vidéo nommé " bobStream ". Inversement, la page " bob.html " est utilisée par Bob. Elle envoie le flux vidéo nommé " bobStream " et reçoit le flux vidéo nommé " aliceStream ". Le code pour Alice est ci-dessous :

```
<html>
 <head>
   <script  src="/[chemin  vers  apideo]/apideo/apideo.js">
</script>
   <script type="text/javascript">
   function init() {
     myConnection = Apideo.connect("112233445566778899")
     myRoom = myConnection.joinRoom("myroom")
     myRoom.startCamera("cameraDiv","streamAlice", {width:320,
 height:240})
     myRoom.playStream("playerDiv","streamBob",  {width:320,
 height:240})
   }
   </script>
 </head>
```

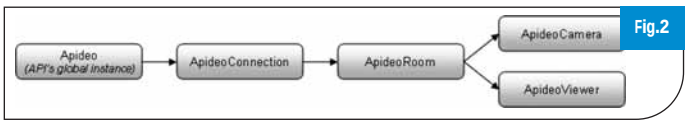


Fig.2

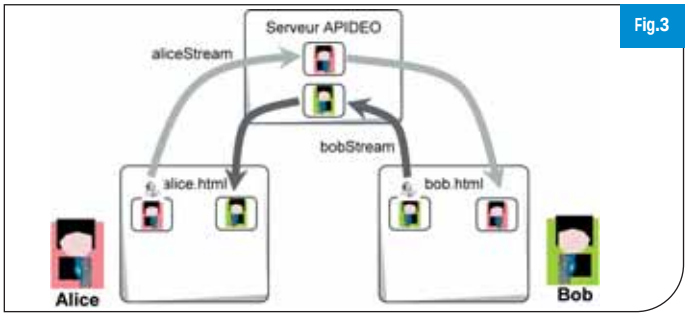


Fig.3

```
  <body onload="init()">
    <h1>Alice</h1>
    <div id="cameraDiv"></div>
    <div id="playerDiv"></div>
  </body>
</html>
```

Nous commençons par inclure le fichier apideo.js et créons 2 éléments " div ". L'un accueillera l'image provenant de notre webcam, l'autre l'image de notre interlocuteur. La fonction *init* contient toute la logique de l'application. La méthode Apideo.connect prend en paramètre la clef que vous avez obtenue à votre inscription sur le site. Dans cet exemple, vous devrez donc remplacer "112233445566778899 " par votre clef. La fonction retourne une " ApideoConnection ". Nous utilisons cette " ApideoConnection " pour nous connecter à la room nommée " myroom ". Dans Apideo, il n'y a pas de création explicite de " room ". L'utilisateur rejoint une room, simplement. Si la room n'existe pas, elle est créée par le service. Lorsque tous les utilisateurs quittent une room, celle-ci disparaît. Une fois la connexion établie, nous rajoutons la caméra, ainsi que le flux joué. La méthode *startCamera* démarre la capture du flux vidéo à partir du PC de Alice. Alice va pouvoir voir son image qui va apparaître dans l'élément dont l'id est " cameraDiv ". Ce flux va être envoyé sous le nom " streamAlice " au serveur. Le troisième paramètre spécifie la largeur et la hauteur du flux. Il existe de nombreux paramètres permettant de régler la résolution de la caméra et le taux de compression du flux. La méthode *playStream* récupère le flux nommé " bobStream " à partir du serveur et l'affiche dans l'élément dont l'ID est " playerDiv ". Le code de Bob sera symétrique. Il suffit de reprendre la page d'Alice et de modifier les 2 dernières lignes de la fonction *init* ainsi :

```
myRoom.startCamera("cameraDiv","streamBob", {width:320, height:240})
myRoom.playStream("playerDiv","streamAlice", {width:320, height:240})
```

Envoyer des messages

On peut aussi échanger des messages entre les utilisateurs connectés à une même " room ". Cette caractéristique est très pratique pour implémenter un chat par exemple. Dans cet exemple, nous allons modifier notre application pour partager un champ texte entre Alice et Bob. Dès qu'Alice ou Bob taperont un caractère dans la zone de texte, celle-ci sera synchronisée sur l'autre poste. Il s'agit d'un prototype de formulaire partagé entre utilisateurs. L'envoi des évènements dans Apideo se fait avec le design pattern " Observer " : les messages sont publiés vers une " room ". Les messages sont typés. Chaque utilisateur peut s'abonner pour une room à un type de message pour les recevoir. L'abonnement à un type de message s'effectue par la fonction *registerListener* et l'envoi de message par la fonction *sendEvent*.

```
<html>
  <head>
    <script src="/[chemin vers apideo]/apideo/apideo.js"></script>
    <script type="text/javascript">
    function init() {
      myConnection = Apideo.connect("112233445566778899")
      myRoom = myConnection.joinRoom("myroom")
      myRoom.startCamera("cameraDiv","streamAlice", {width:320,
  height:240})
      myRoom.playStream("playerDiv","streamBob", {width:320, height:
240})

      myRoom.registerListener(changeField, "onUpdate")
    }

    function changeField(msg) {
      document.getElementById("shared").value=msg
    }

    function sendUpdate(msg) {
      myRoom.sendEvent(msg, "onUpdate", true)
    }
    </script>
  </head>
  <body onload="init()">
             <h1>Alice</h1>
    <div id="cameraDiv"></div>
    <div id="playerDiv"></div>
    <div>
      <label for="shared">Shared field:</label>
      <input  type="text"  id="shared"  name="shared"  onkeyup
="sendUpdate(this.value)" />
    </div>
  </body>
</html>
```

A chaque pression d'une touche, la fonction *sendUpdate* s'exécute. Celle-ci appelle *myRoom.sendEvent* qui envoie l'évènement. *sendEvent* prend 3 paramètres. Le premier paramètre est le message. Il peut s'agir de n'importe quel objet JSON. Dans notre cas, ce sera une simple chaîne de caractères. Le deuxième message est le type de message. Ici : " onUpdate ". Le troisième paramètre est un booléen permettant d'éviter les effets d'écho. Il garantit que l'utilisateur envoyant un message ne le recevra pas. A l'initialisation, nous enregistrons un "listener" sur les messages de type "onUpdate". La fonction *myRoom.registerListener* accepte 2 paramètres. Le premier paramètre est la fonction Javascript qui sera appelé à la réception du message. Le second est le type de message écouté. Dans notre application, chaque message de type " onUpdate " reçu déclenchera l'exécution de la fonction *changeField.* Celle-ci reçoit en paramètre le message qui a été envoyé. Elle met alors à jour le champ texte de l'utilisateur.

Conclusion

Nous avons vu en quelques minutes comment rajouter une conversation vidéo sur un site web, et comment partager un formulaire entre plusieurs utilisateurs. Il ne s'agit évidemment que d'un début qui nécessiterait bien des modifications. Par exemple, les utilisateurs n'ont pas forcément tous des webcams. Il faut alors leur offrir un module de chat. Nous pourrions aussi régler de manière fine les paramètres du flux vidéo, ou encore modifier l'application pour qu'elle accepte plusieurs utilisateurs. Apideo propose des API pour ces différents usages. Apideo ne propose que les briques de base de la communication entre navigateurs. Co-browsing, vidéoconférence, suivi des utilisateurs connectés à votre site web… libre à vous d'inventer les usages que vous souhaitez !

■ David **Négrier**
Directeur technique de The Coding Machine

Epitech
Le réseau et la passion

L'Epitech se veut une école ouverte à tous les passionnés, prêts à travailler dur. Pour coller au mieux aux besoins actuels des entreprises, elle privilégie le travail en équipe, sur des projets, et l'interculturalité.

Nicolas Sadirac, directeur de l'Epitech

" Transformer la passion de l'informatique en une expertise durable et reconnue ", tel est le leitmotiv d'Epitech, la " petite sœur " d'Epita, appartenant au même groupe (Ionis, cf. Programmez! n°114). Deux écoles qui forment des informaticiens en 5 ans après le bac. Mais, alors que l'Epita insiste sur la largeur de champ de connaissances, Epitech se caractérise par la profondeur de connaissances. Et contrairement aux élèves de la première, qui ne se spécialisent qu'en dernière année, en choisissant une option, les élèves de l'Epitech connaissent des techniques particulières dès le début, puisque l'essentiel du cursus se fait sous forme de projets. Ce n'est qu'à partir de la 4e année qu'ils prendront du recul pour pouvoir assurer la fonction de chef de projet, avec un aspect managérial et une démarche qualité. La démarche, comme la pédagogie de l'Epitech, se résument en une phrase : *" Former autour des projets, où l'étudiant est actif dès le début ",* explique **Nicolas Sadirac,** directeur de cette école d'informatique. Une démarche que l'Epitech est en train de décliner au plan régional comme international.

Sébastien Benoît, directeur d'Epitech Paris

Epitech essaime

Lancé il y a 3 ans, le mouvement vers l'international se développe. La maîtrise de l'anglais s'inscrit dans ce mouvement. La langue est abordée sous forme de cours, travaux dirigés, groupes de conversation, e-learning, stages linguistiques. Cet enseignement s'adapte au niveau de l'élève : uniquement cours et TD s'il est débutant, uniquement conversation s'il maîtrise la langue. Avec réévaluation tous les 2 mois et un niveau minimum exigé pour passer d'une année à l'autre. A partir de la 4e année, tout se fait en anglais. D'autant que l'élève passe la 4e année à l'étranger, de préférence dans un pays anglophone (Etats-Unis, Canada, Australie, Inde...), mais aussi Chine ou Russie, où les élèves apprennent la langue du pays. *" L'objectif est d'acquérir une expérience du travail avec des étrangers, la compréhension de l'international et l'interculturalité ",* commente Nicolas Sadirac. Cette internationalisation se traduit aussi par des partenariats avec des universités, par exemple des cours de business à Berkeley, et des équivalences de diplôme avec des universités.

Soucieuse de développer un réseau, l'Epitech a aussi une stratégie d'implantation d'écoles dans divers pays pour des étudiants locaux. Des écoles existent déjà au Sénégal et au Maroc, une Epitech sera mise en place en Chine en février 2009 et à Bangkok en septembre 2009, avec pour chaque établissement un responsable et un directeur du développement. A l'instar de Paris, dont l'école est dirigée par **Sébastien Benoît**.

Un autre axe de développement du réseau, c'est l'ouverture d'établissements en province : l'année dernière a vu l'ouverture de 6 établissements, cette année, il y en a 9 et l'an prochain 11. Il y a aujourd'hui 850 nouveaux étudiants en France, dont 500 en province. Comme pour l'école parisienne, les écoles de province font la 4e année à l'étranger et la 5e à Paris. *" Nous voulons que les élèves soient ensemble : en juin, tous les élèves de France sont réunis à Paris ",* indique Nicolas Sadirac.

Le sens du projet

" 100% de l'évaluation et 100% de l'apprentissage sont basés sur des projets ", rappelle Nicolas Sadirac. Une stratégie gagnante puisque 100% des élèves sont en entreprise à la fin des études. Cela va de pair avec la spécificité du recrutement des élèves : l'Epitech n'hésite pas à admettre *" un élève qui ne sait rien au départ, mais qui est capable de travailler en équipe et de faire émerger la connaissance, peut apporter de la plus-value à l'entreprise. "* En effet, les admissions se font après bac S-STI, avec encadrement spécifique pour les passionnés d'informatique des autres bacs. Une fois entré dans l'école, c'est là que tout commence pour l'étudiant : *" On travaille dur, mais pas de devoirs à la maison, tout se fait à l'école. Le taux d'abandon est de 20% sur 5 ans, dont plus de 15% sur la première année, parce que c'est trop dur. " " On ne peut pas tout enseigner, mais il y a un vrai besoin de capacité d'évolution, un renouvellement des*

connaissances bien plus rapide que dans les autres métiers ", ajoute le directeur de l'Epitech. Contrairement à un enseignement magistral, l'approche par projet permet à l'élève de trouver lui-même la connaissance heuristique. *" L'heuristique est propre à chacun, on ne l'enseigne pas. En partant d'une idée de projet, l'étudiant est ainsi censé faire émerger la théorie. "* Plutôt que des compétences techniques, on apprend l'organisation et la structure de projet. De plus, dans un projet, chacun arrive à trouver sa place, avec ses qualités.

Une exigence : la qualité et la capacité à se jauger. Ainsi, un projet limité à 80% d'une application et qui réalise effectivement ces 80% est jugé bon, alors qu'un projet visant 100%, mais ne réalisant que 95% est refusé. L'élève doit aussi planifier son travail et aboutir à une adéquation entre ce qu'il imagine et ce qu'il arrive à faire effectivement. La capacité à déterminer le temps doit être acquise au niveau individuel dès la première année, et en équipe la 2e année. *" Ce sont des valeurs très importantes dans un projet ",* insiste Nicolas Sadirac.

De la passion à la recherche

Autre spécificité de l'Epitech : l'organisation en laboratoires pédagogiques et de recherche. Pédagogique, car le labo dispense un enseignement dans l'école ; recherche, avec ouverture vers l'extérieur. Par exemple, le LSE (Laboratoire Système et Sécurité de l'Epitech) a une alliance avec une université dans le Nebraska. A la différence de Supinfo, *" l'Epitech n'a pas de labos captifs ; nous ne faisons pas de certification, qui est trop orientée connaissance instantanée, même si les élèves ont la possibilité de passer des certifications ",* précise Nicolas Sadirac. Enfin, pour compléter cet enseignement très technique, Epitech a lancé un nouveau module " expression écrite et orale ". *" Les élèves rédigeaient déjà des rapports de stage, des cahiers des charges, des cahiers de recette, mais cela se faisait surtout par copier-coller ",* explique Nicolas Sadirac. Désormais, dès la première année, ils suivent des cours de rédaction technique. Une matière obligatoire et éliminatoire.

L'Epitech a pour ambition de transformer des passionnés en experts, qu'ils soient développeurs, journalistes, marketeurs, graphistes, etc. *" Nous voulons aller vers ceux qui valorisent la technologie ",* conclut Nicolas Sadirac. *" Dans un contexte d'évolution, le savoir-être est plus important que le savoir, qui est très vite dangereux, car obsolète et enfermant. "*

■ Claire **Rémy**

EPITECH: un campus de plus de 2000 étudiants

- Création en 1999 à Paris
- Cursus de 5 ans en 2 cycles après le bac :
  - 1er cycle en 3 ans, aboutissant au titre de " Bachelor Epitech en technologies de l'information " (à Paris ou en régions)
  - 2e cycle en 2 ans, dont les cours sont dispensés en anglais (4e année à l'international, 5e année à Paris), aboutissant au titre Epitech d' " Expert en technologies de l'information " homologué par l'Etat niveau 1 (CNCP)
- 250 professeurs / intervenants
- 2050 étudiants sur le campus
- 8 laboratoires pédagogiques et de recherche, qui donnent lieu à des articles (40 parutions dont 15 publiées dans des revues scientifiques de renommée internationale)
- 12 écoles en France : Bordeaux, Lille, Lyon, Marseille (ouverture rentrée 2009), Montpellier, Nice, Nancy, Nantes, Paris, Rennes, Strasbourg, Toulouse
- 3 écoles hors de France : Epitech Chine, Epitech Maroc, Epitech Sénégal
- Plus de 100 universités et instituts partenaires dans 43 pays
- Plus de 2000 entreprises partenaires
- Le réseau Epitech : 30 salles informatiques de 20 à 110 postes, totalisant plus de 2500 postes de travail, disponible 24h / 24, 7 jours / 7
- Site : www.epitech.eu

Les EPITECH LABS

LSE

Le LSE, Laboratoire Système et sécurité de l'EPITECH, constitue le pôle de développement high-tech de l'école.

L'une des activités principales du LSE, spécialisé dans la Recherche et Développement, est de réaliser des missions ponctuelles pour les entreprises partenaires.

GAMEDEV'LAB

Au sein du Gamedev'Lab, les étudiants ont la possibilité de mettre à profit leurs compétences et d'acquérir les notions de base du développement de jeux vidéo sur support DS, XBox 360 et PC.

Pour aider les étudiants à mieux appréhender les besoins réels de l'industrie, de nombreux partenariats ont été conclus depuis 2007. Parmi eux, UBISOFT.

LABEIP

Le LABEIP, Laboratoire EPITECH Innovative Projects, est l'entité d'EPITECH qui prend en charge la logistique, la communication et la promotion des " EPITECH Innovative Projects ".

LERIA

Le LERIA, Laboratoire de Recherche en Informatique Appliquée, se veut autonome, tout en intégrant le cursus d'enseignement de l'EPITECH et en proposant des extensions qui intéressent ce dernier. Les principaux axes abordés par les membres du LERIA sont l'intelligence artificielle, les réseaux de neurones, les biotechnologies, la sécurité et la vérification formelle.

LAB'FREE

De création récente, ce laboratoire réunit des étudiants, des associations, des entreprises et des institutions autour de logiciels libres.

ESL

L'ESL, EPITECH Security Lab, est une structure pédagogique et un laboratoire de recherche de l'EPITECH, offrant aux étudiants la possibilité d'appréhender la conception de " codes malicieux autoreproducteurs ", ainsi que les principes des attaques et défenses au niveau applicatif et réseau.

LAB'ELEC

Le Laboratoire Electronique d'EPITECH propose aux étudiants, comme aux entreprises, son savoir-faire en matière d'électronique numérique (microprocesseur, microcontrôleur et FPGA) et de mécatronique (alliance de l'informatique, l'électronique et la mécanique).

LAB'RÉSEAU

L'enseignement des manipulations et de la configuration du matériel Cisco et du réseau en général est assuré par le Lab'Réseau.

Travailler en freelance

1re partie

Les informaticiens ont la possibilité de travailler en freelance. Parfois subi, plus souvent choisi, ce statut n'est pas toujours bien connu. A partir d'enquêtes et de témoignages, nous faisons le point sur cette manière d'exercer le métier d'informaticien.

Le statut de freelance s'est particulièrement développé avec l'essor d'internet, l'informatisation des très petites entreprises et le développement des applications domestiques. *" Le travail en freelance est en très forte progression : supérieure à +10% ",* estime Michel Paysant, de Freelance en Europe.

Samuel Breton, prestataire de création de sites internet, qui a créé sa société, Lexik

Pourquoi ?

Le choix d'indépendant est généralement un choix personnel. Pour pouvoir se consacrer à sa passion, la création de logiciels, le développement de sites web, etc., être libre de choisir ses projets, ses clients, voire son lieu et son temps de travail. *" Nous avons préféré créer notre emploi et être libre de nos choix stratégiques plutôt que d'avoir un patron ",* affirme **Samuel Breton,** prestataire de création de sites internet, qui a créé sa société, Lexik (www.lexik.fr).

Parfois, c'est aussi tout simplement parce que l'employeur ne propose plus de travail. *" J'ai quitté mon dernier employeur vers l'âge de 45 ans et, après un séjour à l'étranger, je n'ai plus retrouvé de travail en France ",* raconte Nicolas Payelle. Le statut d'indépendant lui a permis de retrouver une activité.

Le télétravail – qui est quasiment la règle chez les freelances – peut aussi être un argument de choix. Mais il ne faut pas se faire d'illusion : le freelance travaille beaucoup, comme en témoigne Annie Garcia-Dutaitre (Sekhy Informatique Conseil), indépendante depuis 10 ans : *" Il ne faut pas choisir ce statut pour l'argent. C'est l'envie de créer, de construire, qui m'a amenée à l'indépendance. Comme la plupart des informaticiens, le freelance ne compte guère son temps ; on travaille plus de 10 heures par jour, plutôt 45 à 57 heures par semaine, souvent tard le soir, le week-end, et peu ou pas de vacances. "*

Le freelance travaille seul dans la plupart des cas, parfois en équipe avec d'autres indépendants. La première question qu'il doit se poser est : " Comment en vivre ? " Avoir une activité régulière est parfois difficile au début. Il court aussi le risque de ne pas être payé ou insuffisamment, de sous-estimer le délai et donc le coût de certains projets, notamment lorsque le client demande des compléments en cours de projet. Toutes difficultés qu'il doit assumer seul.

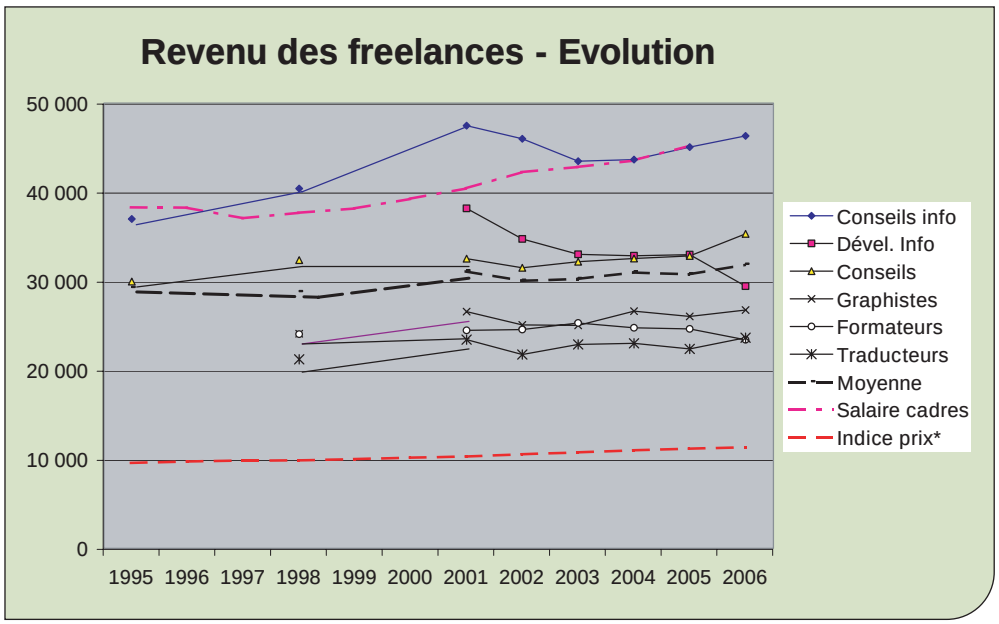
Cependant, être développeur freelance ne veut pas dire être isolé au monde. Il existe, d'une part, des réseaux et sites de mise en relation entre prestataires et donneurs d'ordre, qui aident à trouver des affaires et peuvent éventuellement servir aussi pour le paiement, moyennant commission ; d'autre part, des sociétés de portage qui n'ont généralement pas d'activité commerciale, mais se contentent de facturer au client et de payer un salaire au prestataire ; là aussi, une commission reste acquise à l'intermédiaire.

Le développeur freelance, un métier aux multiples facettes

Pratiquement tous les métiers de l'informatique peuvent s'exercer en freelance, mais de préférence de petits projets, qui peuvent être réalisés par une personne en quelques jours. Les indépendants sont aussi souvent sollicités pour les sites web (développement, aspects graphiques, webmastering), le conseil et la formation. Le freelance doit réunir au moins trois activités :

- développement proprement dit, ce qui est son métier ;
- gestion de projets, d'affaires... ;
- commercial, souvent ressenti comme la partie la plus gourmande

Evolution des revenus des freelances Source : Freelance en Europe



Type de projets et métiers les plus fréquents des freelances :

- Projets web et multimédia - Sur Codeur, 70% des projets sont de type web (développement de site, refonte, amélioration), référencement, graphisme, traduction...
- Formation chez le client
- Services à la personne et aux TPE

en temps et pour laquelle le développeur est rarement bien préparé.

De plus, le freelance doit prendre le temps de se former pour rester toujours à la pointe de la technologie, ce qui est sa valeur ajoutée. Or ce ne sont généralement pas les donneurs d'ordre (ni d'ailleurs les SSII intermédiaires) qui paient la formation. *" Idéalement, le choix d'un freelance présuppose que ce dernier maîtrise les compétences nécessaires à l'aboutissement du dossier confié. Il peut cependant nous arriver de compléter, à nos frais, et si le pré-requis minimal est présent, certains points de connaissances inhérents au degré de technicité d'un dossier ",* indique un donneur d'ordre. *" Au moment où l'on passe le cap de freelance, il faut déjà avoir de la bouteille : le client attend beaucoup de nous. L'indépendant travaille sans filet, il n'a pas droit à l'erreur, il ne compte pas ses heures ",* raconte **Sami Jaber,** consultant et architecte spécialisé en .net, qui a créé sa société (www.dng-consulting.com) il y a un peu plus d'un an.

La démarche commerciale

C'est l'aspect commercial qui représente la principale difficulté du freelance. Son activité nécessite en effet qu'il " se vende " aux clients. *" La crise économique porte en premier lieu sur les freelance ",* note Michel Paysant. *" Les grandes entreprises nettoient leur fichiers fournisseurs, et il est plus difficile aux freelances de se faire référencer. "* Samuel Breton avoue consacrer un tiers de son temps pour la réponse aux appels d'offres, la séduction de clients. *" La démarche commerciale est le point faible des informaticiens indépendants. Ce sont d'abord des techniciens ",* ajoute Michel Paysant. *" L'idéal est de travailler directement avec ses clients. En réalité, les freelances passent souvent par l'intermédiaire de SSII, certains travaillent de manière mixte. "*

Certains freelances sont organisés pour confier cette tâche à un tiers. Pour Sébastien Develay, développeur web et infographiste (www.sdgraphik.com), *" la prospection de clients se fait essentiellement avec le bouche-à-oreille. Pour les tâches de prospection, je travaille également avec un apporteur d'affaires rémunéré à la commission. Pratiquement, je ne consacre donc aucun temps à la prospection. "*

D'ailleurs, une fois leurs compétences reconnues, les freelances n'ont plus trop d'effort commercial à faire. *" Notre choix s'effectue sur le niveau de technicité et l'expérience du futur prestataire ",* souligne un donneur d'ordre qui fait couramment appel aux prestataires freelance.

Les indépendants sont rarement payés au temps passé, ils proposent un coût forfaitaire après étude du dossier. Mais il n'y a pas de règle en la matière. *" Nous facturons au temps passé, qui est estimé lors de la mise en place du cahier des charges. Les forfaits sont plutôt pour les maintenances après livraison ",* précise Samuel Breton. *" Il est aussi possible de faire des interventions 'one shot', facturées à la prestation. "*

Claire **Remy**

Suite de cet article le mois prochain.

Répartition des projets

Projets sur le site Codeur.com

• Graphisme	7%
• Web	16%
• Développement	20%
• Webmastering	24%
• B-to-B	33%

Compétences les plus demandées

• PHP, MySQL	43,7%
• Site clés en mains	20,8%
• CSS / HTML / XML	16,6%
• Site e-commerce	13,5%
• Javascript / Ajax	12,8%
• CMS	10,8%
• Bases de données	10,1%
• Charte graphique	10,1%
• Développement	8,9%
• Flash	8,5%
• ASP, .NET	8,3%
• Java	7,6%
• Référencement	6,4%
• Logo	6,4%
• C / C++	6%
• C#	5,3%

(source : Codeur, 2007)

Sami Jaber, consultant et architecte spécialisé en .net, qui a créé sa société, DNG Consulting (" Design New Guides " en anglais ou " Développer de Nouveaux Guides ")

Revenu annuel moyen des freelances

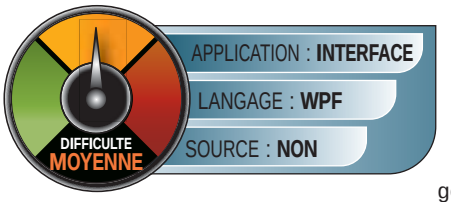
Les chiffres 2006 / 2007 montrent une progression du bénéfice supérieure à +5% pour le conseil et proche de +15% pour la réalisation.

Métier	Recettes nettes 2006	Recettes nettes 2007	Bénéfice en % 2006	Bénéfice en % 2007	Bénéfice 2006	Bénéfice 2007
Conseil en systèmes informatiques	77 344 €	81 187 €	59,9%	60,0%	46 317 €	48 687 €
Réalisation et création de logiciels	52 166 €	56 831 €	56,4%	59,3%	29 440 €	33 718 €

(Source : Freelance en Europe ; UNASA)

Un menu original : le Pie Menu !

Le Pie Menu est un menu particulier, où les différents éléments sont répartis uniformément sur un disque. Sa forte composante graphique permet à son utilisateur d'apprendre à situer facilement ses différents éléments, et de reproduire, très vite, une série de manipulations. Ils peuvent être utiles comme menus contextuels, lorsque des actions répétitives sont nécessaires. Couplés à un petit moteur de reconnaissance de gestes, ils peuvent être particulièrement adaptés à une interaction au stylet. [Fig.1]



Cet article présente les grandes lignes d'une implémentation simple du Pie Menu en WPF (c'est-à-dire sans reconnaissance de gestes). Sa lecture nécessite de posséder de bonnes connaissances en WPF, notamment sur les concepts suivants : Templating et layouting, propriétés de dépendances et propriétés attachées. Le code source étant assez long, vous ne le trouverez pas dans cet article, mais en intégralité sur le site.

CHOIX DE CONCEPTION

Par essence, notre PieMenu n'offre pas de fonctionnalités qui ne soient couvertes par le contrôle Menu de base : Concepts, logique et états restent les mêmes et seule la représentation change. Cette constatation nous permet d'envisager la définition du Pie Menu comme une personnalisation des ControlTemplates d'un Menu et de ses éléments, les MenuItem.

STRUCTURE DU TEMPLATE DE MENU ET MENUITEM

Un Menu WPF est composé de MenuItems. Un MenuItem peut contenir, lui aussi, d'autres MenuItems, définissant ainsi un sous-menu. Les MenuItems possèdent un autre contenu, le Header, qui est généralement une chaîne de caractères.

Comme tout contrôle WPF, l'aspect graphique du Menu est défini par son ControlTemplate. L'élément pilier de celui-ci est l'ItemPresenter : Ce composant a pour rôle d'accueillir, au moment de la résolution de l'Arbre Visuel, l'ensemble des représentations des MenuItems. Utilisant par défaut un StackPanel, son layout peut être changé via la propriété Menu.PanelTemplate.

Les ControlTemplates des MenuItems présentent aussi un ItemPresenter pour leurs sous-éléments. De plus, ils possèdent deux ContentControl : Le premier contiendra le Header, alors que le second sera associé à l'icône du MenuItem.

Notons que le Style appliqué par défaut au MenuItem contient un trigger conditionnant le ControlTemplate à utiliser en fonction du Role de MenuItem (élément de niveau supérieur avec fils, d'un sous-menu sans fils,...). Ainsi, on dénombre un total de 4 ControlTemplates de MenuItems.

SOLUTION ENVISAGÉE

Comme WPF n'offre aucun mécanisme standard pour afficher des éléments définis de manière radiale, nous avons mis en place un mécanisme très simple pour y parvenir.

Nous avons donc défini principalement deux types d'éléments :

- Un type de layout particulier (RadialPanel), dont le rôle est de réorganiser, de manière radiale, ses sous-éléments. Pour cela, il définit deux propriétés attachées RadialPanel.AngleMin et RadialPanel.AngleMax, précisant le secteur angulaire d'un élément. Au rendu, le RadialPanel configurera ces propriétés pour chacun de ses fils, de manière à les distribuer autour du disque. Ces propriétés seront définies comme " read-only ", pour que seul le RadialPanel puisse les modifier. Notez qu'un élément ainsi configuré aura à sa charge d'impacter son affichage en fonction du secteur angulaire qui lui aura été défini.
- Un type de ContentControl particulier (PieContentControl), qui peut représenter n'importe quelle tranche angulaire définie par ses propriétés AngleMin et AngleMax, tout en permettant au designer d'en établir finement le graphisme.

Nous arrivons au passage subtil ;) Nous précisons au Menu d'utiliser un PanelTemplate de type RadialPanel. Les éléments du menu seront donc fils de notre layout personnalisé, et RadialPanel configurera les propriétés RadialPanel.AngleMin et RadialPanel.AngleMax des différents MenuItems. Pour que les MenuItems s'affichent cor-

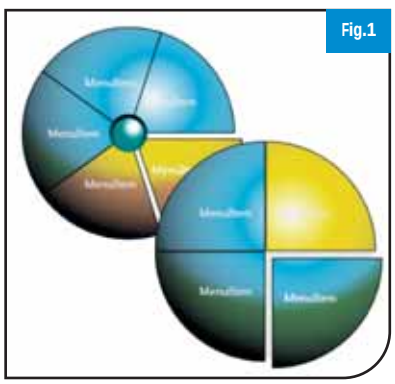


Fig.1

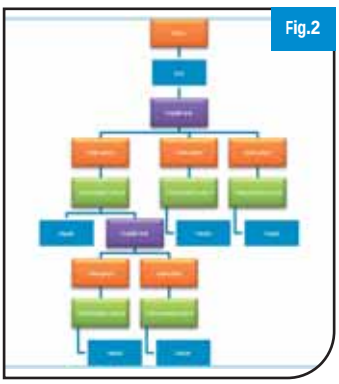


Fig.2

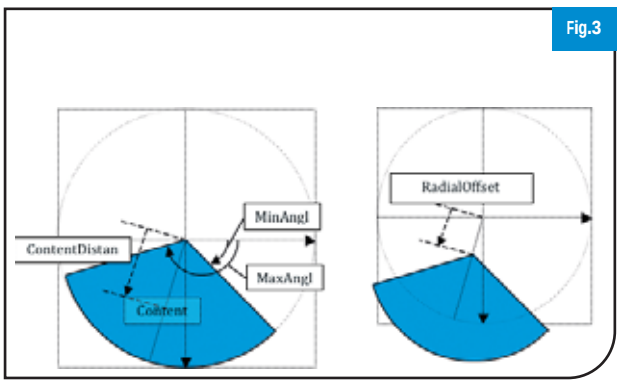


Fig.3

rectement sous leur forme radiale, nous leur appliquons un ControlTemplate principalement basé sur le PieContentControl. Nous lions alors, par Databinding, les propriétés RadialPanel.AngleMin et RadialPanel.AngleMax du MenuItem avec les propriétés AngleMin et AngleMax du PieContentControl de son template ! Le schéma suivant représente une version simplifiée d'un possible arbre visuel obtenu avec notre structure : [Fig.2]

RADIALPANEL

Créer un layout en WPF est assez simple : Il faut pour cela dériver de la classe Panel, et implémenter deux méthodes particulières : MeasureOverride, qui détermine sa taille optimale en fonction des tailles souhaitées de ses fils, et ArrangeOverride, qui positionne ceux-ci et calcule la taille effective.

Dans notre cas, RadialPanel se contente d'étendre ses fils sur l'ensemble de sa surface. Cependant, lors de l'appel à ArrangeOverride, nous recalculerons les secteurs angulaires de chacun de ses fils, et fixerons leurs propriétés attachées.

```
protected override Size ArrangeOverride(Size finalSize)
{
  if (Children.Count != 0)
  {
    Point ptCentre = new Point(finalSize.Width / 2.0, finalSize.
Height / 2.0);
    double dAngularPace = (double)(360.0 / Children.Count);
    double dCurrentAngle = 0.0;

    foreach (UIElement eltChild in Children)
    {
    eltChild.Arrange(new Rect(0, 0, finalSize.Width, finalSize.
Height));

    eltChild.SetValue(AngleMinKey, dCurrentAngle);
    eltChild.SetValue(AngleMaxKey, dCurrentAngle + dAngularPace);

    dCurrentAngle += dAngularPace;
  }
}
```

C'est tout ! RadialPanel ne fait finalement que peu de choses. Le gros du travail est effectué par le PieContentControl et les ControlTemplates associés.

PIECONTENTCONTROL

PieContentControl dérive de ContentControl. Son rôle premier est donc d'afficher un contenu. Dans notre cas, ce contenu est présenté dans un contexte graphique représentant un secteur angulaire de disque.

A cette fin, nous partons du principe que le ControlTemplate du PieContentControl contient l'intégralité du graphisme du disque de fond. Si seul un secteur angulaire doit être affiché, alors nous clipperons la fraction de la surface du disque nous intéressant. En procédant ainsi, le graphiste garde l'entière responsabilité du design du disque tel qu'il doit être affiché quand l'ensemble des PieContentControl d'un Pie Menu sont présents.

Fig.4

PROPRIÉTÉS DE PIECONTENTCONTROL

PieContentControl définit un certain nombre de propriétés, représentées sur la figure suivante : [Fig.3]

- MinAngle et MaxAngle décrivent les angles minimaux et maximaux du PieContentControl.
- ContentDistance est un facteur, entre 0.0 et 1.0, figurant l'éloignement du contenu du contrôle par rapport à son centre.
- RadialOffset est un paramètre permettant de configurer une translation qui sera appliquée sur le Pie, selon son axe médian, pour le faire ressortir du menu, au besoin.

PieContentControl ne définit pas explicitement de propriétés RadiusX et RadiusY de l'ellipse : Nous utiliserons simplement la largeur et la hauteur de notre contrôle.

De plus, chacune des propriétés est enregistrée avec le flag AffectsArrange, précisant que toute modification de la propriété provoquera la réorganisation du Control (invocation de ArrangeOverride)

CONTRAT DE TEMPLATE DE PIECONTENTCONTROL

Afin de se lier à son graphisme, le composant PieContentControl définit un contrat de Template, et s'attend à y trouver trois éléments précis :

- PART_PieContent : Un panel à repositionner au centre du secteur angulaire. Il doit contenir le ContentPresenter du PieContentControl.
- PART_PieBorderPath : Un Path à utiliser comme bordure de la forme géométrique du secteur, afin que le designer puisse en configurer l'apparence.
- PART_ClippedPanel : Le Panel principal à clipper selon le secteur angulaire défini. Il doit contenir le graphisme de fond de notre contrôle (idéalement une ellipse), et, éventuellement, le panel PART_PieContent, si nous souhaitons que celui-ci ne déborde pas de son secteur angulaire. [Fig.4]

Muni de ces quelques contraintes, le designer pourra réaliser l'habillage du PieContentControl. Dans notre exemple, nous n'avons pas mis le graphisme directement dans le ControlTemplate du PieContentControl : Nous avons simplement effectué une liaison de donnée entre le Background du Panel PART_ClippedPanel, et PieContentControl.Background. Il suffira de fixer, dans cette variable, une DrawingBrush contenant le graphisme que le designer souhaite mettre en fond. Cela nous évite de multiplier les ControlTemplate pour gérer les différents états des MenuItems.

Comme toujours, les éléments du template sont récupérés dans la méthode surchargée PieContentControl.OnApplyTemplate. Petite subtilité, après sa récupération, PART_PieContent sera automatiquement centré pour en faciliter le positionnement. Pour rappel, voici comment récupérer un élément du ControlTemplate :

```
public override void OnApplyTemplate()
  {
    base.OnApplyTemplate();
    _PartPieContent = Template.FindName("PART_PieContent",
```

```
this) as Grid;
   }
```

MISE À JOUR DU SECTEUR ANGULAIRE

La mise à jour graphique est effectuée dans une méthode dédiée, PieContentControl.updatePie (). Cette méthode sera directement invoquée au sein de ArrangeOverride (). La taille retournée par base.ArrangeOverride () servira de taille de référence au calcul de notre secteur angulaire.

```
protected override Size ArrangeOverride(Size arrangeBounds)
{
      Size finalSize = base.ArrangeOverride(arrangeBounds);
      updatePie(finalSize);
return (finalSize);
}
```

La méthode PieContentControl.updatePie procède par étapes successives :

- Tout d'abord, elle détermine la géométrie de clipping. Ce calcul est délégué à la méthode computePieGeometry (), et renvoie une Geometry de secteur angulaire correspondant aux paramètres de notre Pie. Il procède par combinaisons logiques de formes élémentaires, en utilisant la méthode Geometry.Combine (d'autres méthodes, plus efficaces, existent, mais l'illustration était intéressante). [Fig.5]
- Elle applique ensuite cette Geometry à _PartClippedPanel.Clip. Le graphisme de fond sera alors clippé selon le secteur angulaire souhaité.
- Elle l'applique à nouveau à _PartPieBorderPath.Data. Le Path de " bordure " épousera alors la forme du secteur angulaire.
- Elle configure ensuite la matrice de translation de _PartPieContent, pour le positionner à sa position correcte. Comme nous l'avons centré dans PieContentControl.OnApplyTemplate, il nous suffit de le décaler de manière radiale, selon l'angle médian, à bonne distance.

```
double dOffsetX = Math.Cos(dMidAngle) * dRadiusX * Content
Distance;
double dOffsetY = Math.Sin(dMidAngle) * dRadiusY * Content
Distance;
_PartPieContent.RenderTransform = new TranslateTransform (dOffset
X, dOffsetY);
```

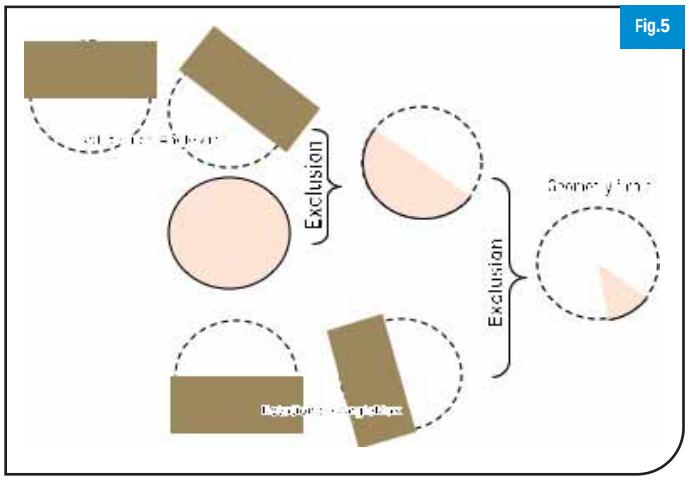


Fig.5

- Finalement, elle applique l'offset RadialOffset, pour que notre tranche se retrouve décalée du facteur désiré – en configurant cette fois-ci la matrice de transformation de l'élément lui-même.

Notre PieContentControl est désormais prêt. Il ne nous reste qu'à définir les templates des Menultem et du Menu.

STYLE ET CONTROLTEMPLATE DES MENUITEMS

Pour les ControlTemplates de Menultems, nous avons décidé de suivre le modèle proposé par défaut : Quatre templates discriminés par un Trigger de Style. Leur principale caractéristique est de se baser sur le PieContentControl plutôt que sur un Border. Nous lions, de plus, les propriétés PieContentControl.MinAngle et PieContentControl.MaxAngle à RadialPanel.MinAngle et RadialPanel.MaxAngle :

```
<PieMenuSample:PieContentControl HorizontalAlignment="Stretch"
Margin="0,0,0,0" Width="Auto" Height="Auto" Template="{Dynamic
Resource
PieContentControlControlTemplate}" MinAngle="{TemplateBinding
PieMenuSample:RadialPanel.AngleMin}" MaxAngle="{TemplateBinding
PieMenuSample:RadialPanel.AngleMax}" Background="{DynamicResource
PieItemNormal}" x:Name="pieContentControl">..</
PieMenuSample:PieContentControl>
```

Nous fixons finalement, dans le style du Menultem, sa propriété ItemsPanel :

```
<Style TargetType="{x:Type MenuItem}" x:Key="MenuItemStyle">
<Setter Property="ItemsPanel" Value="{DynamicResource Radial
ItemsPanelTemplate}"/>
…
</Style>
<ItemsPanelTemplate x:Key="RadialItemsPanelTemplate">
<PieMenuSample:RadialPanel/>
</ItemsPanelTemplate>
```

Enfin, nous ajoutons un trigger sur la propriété Menultem.Hightlight, déclenchant une animation sur PieContentControl.RadialOffset, et le faisant sortir du menu au survol.

STYLE ET CONTROLTEMPLATE DU MENU

Tout comme les Menultems, le style du Menu devra forcer la valeur de Menu.ItemsPanel. Et c'est tout ! Nous avons pour notre part enrichi légèrement le ControlTemplate du Menu pour y ajouter un petit graphisme, au centre de l'ellipse, en superposition des Menultems.

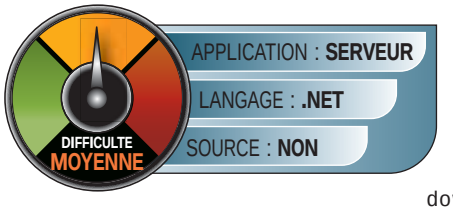
CONCLUSION

Cet article vous a présenté de manière complète comment réaliser un Pie Menu, en jouant sur les Templates de Control existants. Certaines choses restent très perfectibles, notamment l'apparition des Popup des sous-menus, dont la position, non maitrisée, peut nuire à l'utilisabilité. De plus, en l'état, il ne se prête pas idéalement à l'utilisation au stylet. Mais vous avez à présent tous les éléments pour l'améliorer par vous-même !

■ Roland **Tomczak**
Architecte logiciel, concepteur IHM - Intuilab

Développer une application Speech Server avec le Framework .NET

Speech Server 2007 est un serveur de la famille Microsoft Office Communication Server 2007. Il permet de déployer des applications vocales interactives (IVR en anglais), reposant sur le Framework .Net et intégrées à OCS 2007 et aux applications SIP traditionnelles. Le gros avantage de la version 2007 par rapport aux anciennes versions est le support natif de la voix sur IP.



Pour installer Speech Server il faut préalablement avoir mis en place sur le serveur les composants suivants :

• Windows 2003 R2 ou Windows XP. XP permettra sans problème d'utiliser la carte son du PC pour faire du développement.

• IIS et Microsoft Message Queuing (MSMQ).

Pour cette installation, il faut aller dans ajout / suppression de composants Windows et sélectionner l'option Application Server. Ensuite la ligne étant sélectionnée, il faut cliquer sur détails et sélectionner l'option Message Queuing.

- Le Framework .Net 3.0.
- Visual Studio 2005 SP1
- Les extensions de Visual Studio 2005 pour le Windows Workflow Foundation.

Une fois les pré-requis mis en place, on peut lancer l'installation de Speech Server. Lors du choix des composants, pour un serveur de démonstration ou de développement, je recommande d'installer tous les composants.

• Server component
• Development tools
• Documentation
• Administrative tools
• Data Processing Utilities

Pour un serveur de production, seuls sont requis :

• Server Components
• Administrative Tools

Les autres composants ne sont pas nécessaires et il est recommandé de limiter autant que possible le nombre d'éléments installés sur un environnement de production.

Une fois l'installation des binaires effectués, il faut encore mettre en place un ou plusieurs packs de langue. Les packs de langue disponibles avec Speech Server 2007 sont :

French (Canada), French (France), Spanish (United States), Spanish (Spain), English (United States), English (United Kingdom), English (Australia), German (Germany), Chinese (People's Republic of China), Chinese (Taiwan), Italian (Italy), Japanese (Japan), Korean (Korea), Portuguese (Brazil)

Les packs de langue sont téléchargeables depuis cette adresse : http://www.microsoft.com/downloads/details.aspx?FamilyId=BB183640-4B8F-4828-80C9-E83C3B2E7A2C&displaylang=en

MSS est maintenant installé mais non configuré. C'est-à-dire que dans cette configuration il est possible de développer et debugger des applications développées pour Speech Server mais il est impossible de les utiliser dans un environnement OCS.

Si vous l'utilisez uniquement en mode debug, il est alors recommandé d'installer le correctif suivant pour éviter des problèmes d'affichage du designer de Workflow Foundation : http://www.microsoft.com/downloads/details.aspx?FamilyId=4D8B068B-3C45-4EEA-BBC8-C4A4C4201F60&displaylang=en

DÉVELOPPEMENT D'UNE APPLICATION VOCALE INTERACTIVE

Avec Speech Server 2007 et Visual Studio 2005 SP1, il existe 3 technologies pour développer des applications vocales interactives :

• A base de la technologie SALT
• A base de VoiceXML,
• A base de Workflow Foundation de la plate-forme .NET.

C'est sur cette dernière technologie que nous allons nous concentrer dans cet article.

Nous allons mettre en place un exemple simple d'application vocale interactive à base de flux en suivant quelques étapes clés.

• Définir le flux de la conversation
  A cette étape, nous définissons ce que l'application doit dire aux utilisateurs. Soit sous la forme de phrases simples, soit sous la forme de questions / réponses. Ce sont les *Prompts*.

On pourra y ajouter ensuite des activités soit prédéfinies, soit que l'on aura préalablement développées.

• Construire la grammaire
  Pour qu'une application reconnaisse ce que dit l'utilisateur, elle a besoin de connaître quels mots et phrases sont attendus. Construire une grammaire, implique la compilation de listes de réponses que l'utilisateur dira. La construction de cette grammaire doit être pensée assez tôt dans le processus de développement.
• Déployer, gérer et stabiliser l'application

PASSONS AU CAS PRATIQUE

Dans cette démonstration très simple, nous montrerons

1. La manière de créer un Workflow à l'aide du designer
2. Comment créer à l'aide d'activités des messages d'accueil, ainsi que des instructions à exécuter
3. Comment construire une grammaire, et la mettre en relation avec des activités

Dans Visual Studio 2005 SP1 nous avons (après avoir installé les pré-requis) des modèles de développement pour Speech Server (en VB ou en C#). [Fig.1]

1) Pour construire notre première application Vocale Interactive, on choisira ici le modèle " Voice Response Workflow Application ", l'autre étant réservé à la création d'activités réutilisables.
2) Un assistant se charge, nous demandant dans quel langage nous souhaitons notre application. [Fig.2]

A tout moment par code, nous pouvons changer de langue.

3) Une fois les paramètres acceptés, nous nous retrouvons sur une surface de dessin de flux, propre à Speech Server, avec une boîte à outils spécifique. [Fig.3]
4) A présent, nous allons simplement faire dire à notre application " Bonjour et bienvenue sur le répondeur de Programmez " en glissant-déplaçant l'activité de type " statement " à notre flux. [Fig.4]
5) Le tag nous indique qu'il faut ajouter la phrase (le prompt) à dire, en sélectionnant " Edit prompt ", ajoutons la phrase " Bonjour et bienvenue sur le répondeur de Programmez " [Fig.5]
6) Compilons l'application, exécutons là en mode debug en appuyant sur la touche F5.

L'application " Voice Response Debugging Window ", va nous permettre de simuler et d'exécuter notre application. En appuyant sur le bouton " Call ", notre application démarre, et nous devrions entendre, la phrase " Bonjour et bienvenue sur le répondeur de Programmez " [Fig.6]

7) Allons un peu plus loin maintenant, et demandons à notre application de répondre à une question.

Pour ce faire nous allons ajouter l'activité " QuestionAnswer " à notre flux. [Fig.7]

8) Ajoutons notre question " Etes-vous abonné à Programmez ? ", ainsi que les phrases, " Dites OUI ou NON " si jamais un silence s'installe, et " Répondez uniquement par OUI ou par NON " si l'application ne reconnaît pas la commande vocale, comme indiqué ci-dessous. [Fig.8]
9) Maintenant nous allons construire une grammaire très simple qui permettra à Speech Server d'associer ce que l'utilisateur a dit à une action.

Pour cela, il faut ouvrir le fichier ayant l'extension *.gbuilder* comme indiqué sur la figure. [Fig.9]

10) Le panneau suivant s'ouvre. [Fig.10]
11) Pour ajouter des mots clés, il faut procéder comme suit :

Cliquez sur le tag pour ajouter le container " Abonnement ". [Fig.11]

12) Cliquez de nouveau sur le tag pour ajouter les mots clés OUI et NON. [Fig.12]
13) Nous allons maintenant ajouter une réponse que nous validerons. Cliquez sur le tag du panneau " Answers " [Fig.13]
14) Ajoutez les exemples de phrases type réponse, comme indiqué sur la figure [Fig.14]
15) Puis Cliquez sur le tag pour associer nos mots clé. [Fig.15]
16) Il nous reste maintenant à valider nos choix, en appuyant sur le bouton " Parse " comme indiqué sur la figure. [Fig.16]
17) Retournons maintenant sur la surface de dessin de notre flux, et associons à notre activité " QuestionAnwser " notre grammaire. [Fig.17]
18) Pour tester, notre grammaire, nous allons maintenant ajouter une activité de type conditionnel " IfElse " à notre flux [Fig.18]
19) Puis en cliquant sur le tag nous allons créer du code, qui nous permettra de passer dans l'une ou l'autre des branches en fonction de la réponse de l'utilisateur.

Pour la propriété : Condition, il faut choisir " Code Condition ", puis saisir ReponseAbonnement. ReponseAbonnement étant le nom de l'évènement qui sera invoqué. [Fig.19]

20) Ajoutons maintenant du code à notre événement ReponseAbonnement

```
private void ReponseAbonnement(object sender, ConditionalEventArgs e)
    {
```


Fig.1

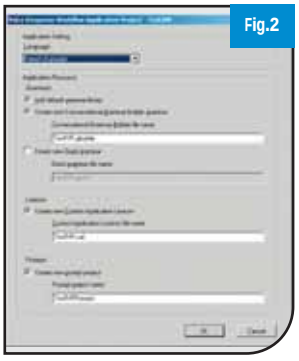


Fig.2


Fig.3


Fig.9


Fig.10

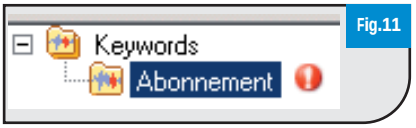


Fig.11

Fig.14

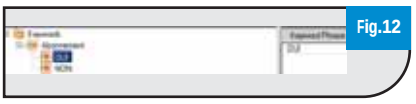


Fig.12

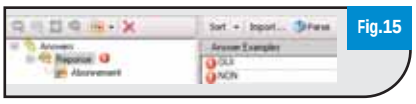


Fig.15

Fig.13


Fig.16

```
      string reponse = this.questionAnswerActivity1.Recognition
Result.Text ;
       if (reponse=="OUI")
                e.Result = true;
       else if (reponse=="NON")
              e.Result = false;
   }
```

Ici nous récupérons dans la variable " reponse ", ce que l'utilisateur a dit, puis nous invoquons à l'aide de la variable " e.Result " l'une ou l'autre des branches. Il est à noter que si l'utilisateur dit autre chose que OUI ou NON (mots clés définis dans notre grammaire), l'évènement ReponseAbonnement n'est pas invoqué. Par contre c'est la phrase " Répondez uniquement par OUI ou par NON " que vous entendrez, comme nous l'avons explicitement indiqué à l'étape 8, au cas ou la commande vocale n'est pas reconnue.

21) Il s'agit maintenant de rajouter à titre d'exemple, deux activités dans les branches ReponseOui et ReponseNon de notre activité " IfElse ". Avec les messages " Vous êtes abonné " et " Vous n'êtes pas abonné " [Fig.20]

TEST ET DÉBOGAGE DE L'APPLICATION

Une fois l'application compilée, nous allons la tester :

1) Pour la démarrer, appuyez sur la touche F5
2) L'outil de simulation " Voice Response Debugging Window " se charge
3) Appuyez sur le bouton " Call " pour initialiser notre application vocale
4) Puisque nous avons une activité de type " QuestionAnswer " dans notre flux, la fenêtre *User Input* apparaît. [Fig.21]

Il est possible de tester soit en rentrant un texte dans la boîte de saisie " Text Input ", mais si vous possédez un bon micro nous allons enregistrer à l'aide du bouton " Start Recording " les réponses

5) Appuyez sur le bouton " Start Recording "
6) Dites " Je ne sais pas "

Une fois la commande enregistrée, soumettez-là à votre application, qui devrait logiquement répondre, " Répondez uniquement par oui ou par non "

7) Sélectionnez l'option " Silence ", puis le bouton " Submit ", vous devriez entendre " Dites oui ou non "
8) Retournez dans le code de votre application à l'évènement *ReponseAbonnement*
9) Mettez un point d'arrêt sur la première ligne
10) Revenez à l'outil de simulation
11) Appuyez sur le bouton " Start Recording "
12) Dites NON
13) Appuyez sur le bouton " Submit "
14) Si la commande a été reconnue, l'application doit s'arrêter en mode débogue.
15) Ensuite, vous devriez entendre le message " Vous n'êtes pas abonné "

CONCLUSION

Nous venons de voir dans notre exemple, qu'avec Speech Server et Windows Workflow Foundation de la plate-forme .NET, il est possible de créer rapidement des applications vocales interactives. Bien évidement dans cet article nous n'avons pas abordé l'ensemble des fonctions, mais sachez qu'il existe des activités complexes, comme la possibilité de transférer des appels, d'invoquer d'autres Flux, d'enregistrer des messages, d'exécuter des services Web, mais également de lever et gérer les exceptions, d'exécuter des boucles, du code en parallèle, j'en passe et des meilleures ! Comme cette technologie est basée sur la plate-forme .NET, la courbe d'apprentissage en est simplifiée.

■ Eric **Vernié** & Damien **Caro**
Microsoft France

Fig.4

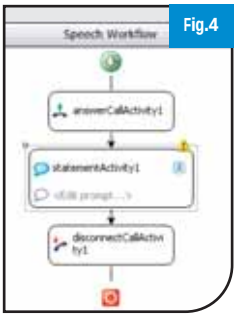


Fig.5

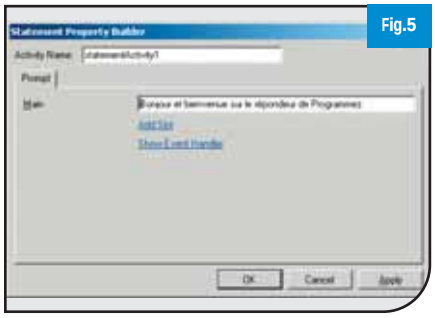


Fig.6

Fig.7

Fig.8

Fig.17

Fig.18

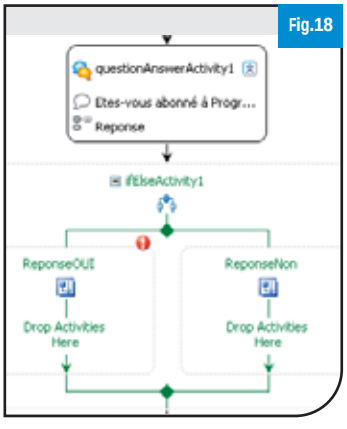


Fig.19

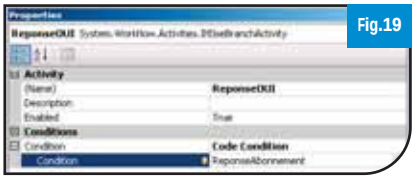


Fig.20

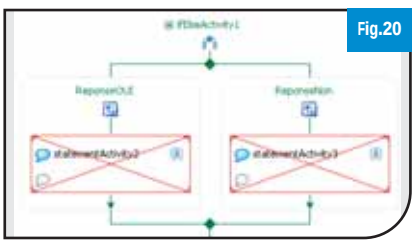
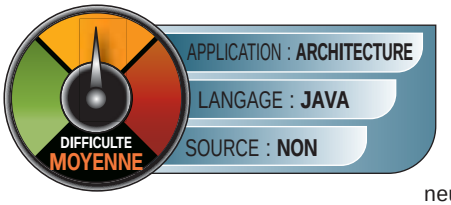


Fig.21

L'architecture et la technologie OSGi

Dans cet article, nous allons décrire les différents concepts de la technologie *OSGi*. Celle-ci permet de développer des applications structurées en composants et de réaliser des architectures orientées service légères. Cette structuration est d'autant plus intéressante que la taille des applications augmente.



Après avoir été longtemps utilisé dans le monde de l'embarqué, *OSGi* a été popularisé par *Eclipse*, son socle reposant sur un conteneur *OSGi* dénommé *Equinox*, il est désormais de plus en plus utilisé dans les développements d'applications " classiques " et serveur. La technologie *OSGi* se caractérise notamment par son utilisation de fichiers *Jar* pour le *packaging* des composants et par l'utilisation d'un mécanisme de cloisonnement de *classloaders*, éléments sur lesquels nous reviendrons tout au long de cet article. Ce dernier aspect est particulièrement appréciable puisqu'il permet aux fournisseurs de composants d'utiliser des bibliothèques *Java* hétérogènes et même incompatibles sans effet de bord entre eux.

Dans cet article, nous allons nous attarder sur les différentes briques du cœur de la technologie.

ARCHITECTURE D'OSGI

Afin d'être le plus portable possible, *OSGi* a fait le choix de la technologie *Java* en tant que plate-forme d'exécution, les différentes briques de son architecture reposant donc sur tous les mécanismes de *Java*. La figure 1 illustre les différentes couches applicatives de son architecture.

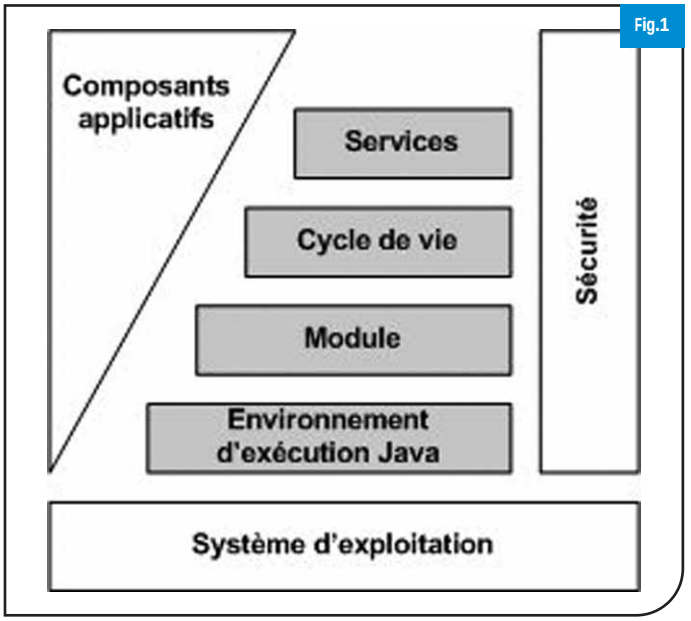
La première couche (*Module*) adresse la gestion des composants et de leurs dépendances, aussi bien au niveau du chargement des classes (*classloading*) que de la gestion de leurs visibilités et de leurs versions.

La deuxième couche (*Cycle de vie*) prend quant à elle en charge les états des composants supportés par le conteneur afin de les gérer, ainsi que les différentes interfaces de programmation correspondantes.

Enfin, la dernière couche (*Services*) offre la possibilité aux composants de mettre à disposition des services au sein d'une même machine virtuelle d'une manière légère, tout en masquant leurs implémentations à leurs utilisateurs.

COMPOSANT OSGI

Avec *OSGi*, un composant (dénommé également *bundle*) n'est autre qu'un ensemble de classes Java *packagées* dans un fichier *Jar* standard. Les propriétés du composant sont spécifiées dans le fichier



MANIFEST.MF du répertoire *META-INF* en se fondant sur des en-têtes spécifiques dont les principales sont les suivantes :

- *Bundle-Name :* le nom du composant;
- *Bundle-Version :* la version du composant;
- *Import-Package :* la liste des packages utilisés par le composant;
- *Export-Package :* la liste des packages du composant utilisables à l'extérieur de ce dernier;
- *Require-Bundle :* la liste des composants nécessaires au fonctionnement du composant;
- *Bundle-Activator :* l'entité d'activation du composant;
- *Bundle-Classpath :* les bibliothèques constituant le *classpath* interne du composant.

Une originalité d'*OSGi* est qu'un composant peut mettre à disposition aussi bien des *packages* que des services afin qu'ils soient consommés par d'autres composants. Exposer des *packages* signifie que toutes les classes contenues dans ces derniers sont accessibles depuis l'extérieur du composant. En effet, un composant correspond à une boîte noire dont rien n'est visible par défaut. Cette isolation est stricte puisqu'elle se positionne au niveau des *classloaders*.

Bien qu'un fichier *Jar* puisse être utilisé directement dans une application *Java*, ce dernier doit être déployé dans un conteneur *OSGi* afin de bénéficier de tous les aspects décrits précédemment. Nous reviendrons sur le déploiement en fin d'article.

Les autres caractéristiques des composants consistent en la gestion de leurs dépendances. Il est en effet possible de spécifier les composants utilisables à l'exécution pour les traitements d'un autre composant, un mécanisme de gestion de versions étant disponible à ce niveau. Notons que ces dépendances peuvent aussi bien consister en des packages que des services.

GESTION DES DÉPENDANCES

Au niveau des *packages*, la configuration des dépendances d'un composant se réalise par l'intermédiaire de l'en-tête *Import-Packa-*

ge, en-tête permettant de spécifier la liste des *packages* utilisés avec éventuellement des informations relatives à la version utilisée. Si aucune information relative n'est spécifiée, la dernière version est automatiquement utilisée.
Ainsi, par exemple, si un composant veut utiliser la bibliothèque *JDom* ainsi que les *API OSGi*, il suffit de spécifier la valeur suivante pour l'en-tête :

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: TestComposant Plug-in
Bundle-SymbolicName: TestComposant
Bundle-Version: 1.0.0
Bundle-Activator: fr.programmez.osgi.TestActivator
Import-Package: org.jdom,
 org.osgi.framework;version="1.3.0"
Export-Package: fr.programmez.osgi
```

A ce niveau, il convient de s'assurer d'avoir déployé dans le conteneur des versions des bibliothèques sous forme de composants *OSGi*. Ces dernières ne sont pas toujours disponibles et il convient alors d'utiliser des outils tels que BND (http://www.aqute.biz/Code/Bnd) afin de transformer les fichiers *Jar* classiques en composants *OSGi*.
D'un autre côté, afin de mettre à disposition des *packages* du composant, l'en-tête *Export-Package* doit être utilisé. Par exemple, si le package *fr.programmez.osgi* d'un composant doit être mis à disposition, l'en-tête suivant doit être spécifié :

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: TestComposant Plug-in
Bundle-SymbolicName: TestComposant
Bundle-Version: 1.0.0
Bundle-Activator: fr.programmez.osgi.TestActivator
Import-Package: org.jdom,
 org.osgi.framework;version="1.3.0"
Export-Package: fr.programmez.osgi
```

INTERFACES DE PROGRAMMATION

Dans certains cas, il est nécessaire d'interagir avec le conteneur, les deux principaux cas étant la spécification de traitements au démarrage et à l'arrêt d'un composant ainsi que la manipulation des services.
Le premier aspect se met en œuvre par l'intermédiaire de l'interface *BundleActivator* et de l'en-tête *Bundle-Activator.* Les traitements doivent être spécifiés dans une implémentation de cette interface, implémentation dont le nom doit être renseigné dans l'en-tête précédent afin que le conteneur *OSGi* puisse l'utiliser. Le code suivant permet d'afficher des messages de traces aussi bien au démarrage qu'à l'arrêt du composant :

```
public class TestActivator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        System.out.println("Démarrage du composant Test");
    }

    public void stop(BundleContext context) throws Exception {
        System.out.println("Arrêt du composant Test");
    }
}
```

Afin que les messages s'affichent, l'en-tête précédemment cité doit être spécifié de la manière suivante :

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: TestComposant Plug-in
Bundle-SymbolicName: TestComposant
Bundle-Version: 1.0.0
Bundle-Activator: fr.programmez.osgi.TestActivator
Import-Package: org.jdom,
 org.osgi.framework;version="1.3.0"
Export-Package: fr.programmez.osgi
```

L'autre entité incontournable des interfaces de programmation *OSGi* est l'interface *BundleContext.* Cette dernière permet véritablement d'interagir avec le conteneur afin de réaliser de nombreuses opérations telles que le référencement de composants, l'enregistrement et le référencement de services, l'installation de composants et la spécification d'observateurs. Cette entité est accessible notamment lors de l'utilisation de l'interface *BundleActivator* décrite précédemment. Nous allons détailler l'utilisation de cette entité dans la prochaine section afin d'enregistrer et d'utiliser des services.

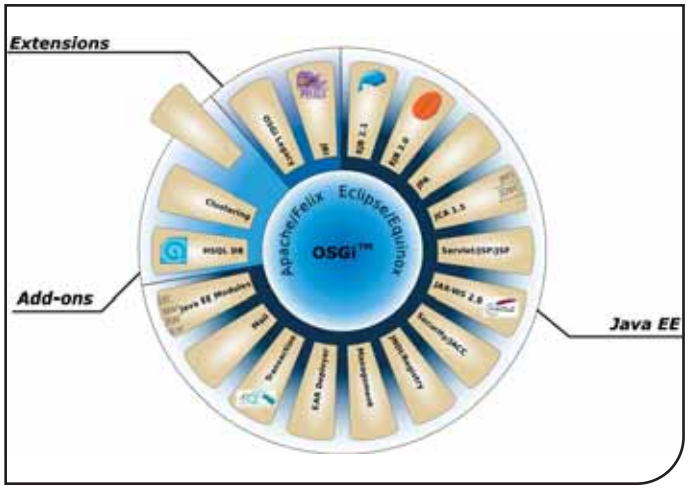
MISE EN ŒUVRE DE SERVICE

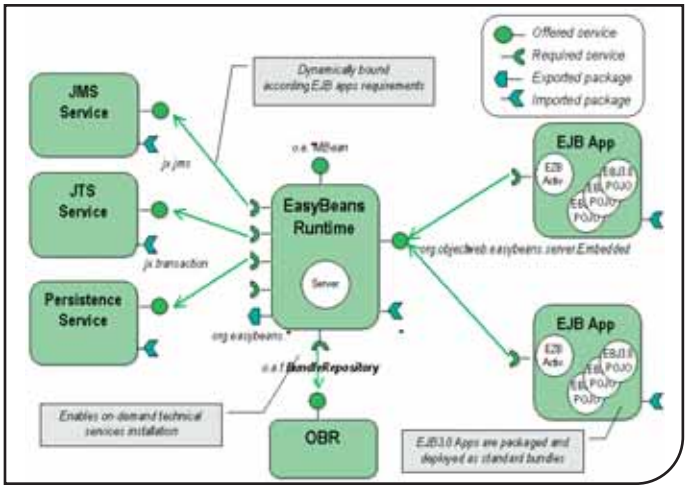
Comme nous l'avons souligné précédemment, *OSGi* offre la possibilité de mettre à disposition et de consommer des services. Au niveau des composants, un service est mis en œuvre par l'intermédiaire d'un *POJO* (classe *Java* simple) en suivant la programmation par interface. En effet, une interface décrit le contrat du service qui doit être implémenté par le *POJO. OSGi* permet, de par son fonctionnement, de complètement découpler ces deux éléments et de masquer l'implémentation à l'utilisateur du service.
L'enregistrement et l'accès à un service se réalise par l'intermédiaire de l'entité *BundleContext* décrite précédemment. A cet effet, cette dernière met à disposition la méthode *register* afin d'enregistrer un service dans le conteneur tandis que la classe *ServiceRegistration* fournit la méthode *unregister* pour le désenregistrement. Le code suivant illustre l'utilisation de ces deux méthodes :

```
    private ServiceRegistration enregistrementService;

    private void enregistrerService(BundleContext context) {
        TestService service = new TestServiceImpl();
        String serviceName = TestService.class.toString();
        enregistrementService = context.registerService(
                                    serviceName, service, null);
    }

    private void deenregistrerService(BundleContext context) {
        enregistrementService.unregister();
    }
```

Notons qu'il est conseillé d'utiliser le nom de l'interface implémentée par le service en tant que nom d'enregistrement.
Par la suite, ce service peut être utilisé par l'ensemble des composants présents dans le conteneur de la manière suivante en se basant sur les méthodes *getServiceReference* et *getService* de l'interface *BundleContext* :

```
    private void utiliserService(BundleContext context) {
        String serviceName = TestService.class.toString();
        ServiceReference serviceReference
                = context.getServiceReference(serviceName);
        TestService service
                = (TestService) context.getService(serviceReference);

        String retour = service.test("Un paramètre");
        System.out.println("Retour: "+retour);

        context.ungetService(serviceReference);
    }
```





Afin que l'interface du service soit visible (ici l'interface *TestService*), il est impératif que le *package* de cette dernière soit importé par le composant utilisant le service.

Notons qu'une extension d'*OSGi* dénommée *Declarative Services* permet de configurer les services d'un composant par l'intermédiaire d'un fichier *XML* sans utiliser les interfaces de programmation d'*OSGi*.

DÉPLOIEMENT ET CYCLE DE VIE

Pour terminer cet article, nous allons décrire la manière de déployer un composant dans un conteneur *OSGi*. Plusieurs conteneurs sont disponibles gratuitement sur Internet, les principaux étant *Felix* (hébergé par *Apache*), *Equinox* (utilisé dans la plate-forme *Eclipse*) et *Knopflerfish*.

Le déploiement d'un composant se réalise par l'intermédiaire des outils d'administration du conteneur, la plupart du temps, un outil en ligne de commande et éventuellement une interface *Web* étant mis à disposition à cet effet. Le code suivant décrit l'installation et le démarrage d'un composant *OSGi* en ligne de commande avec le conteneur *Felix* :

```
-> install file:///home/templth/osgi/programmez/test.jar
Bundle ID: 5
-> start 5
->
```

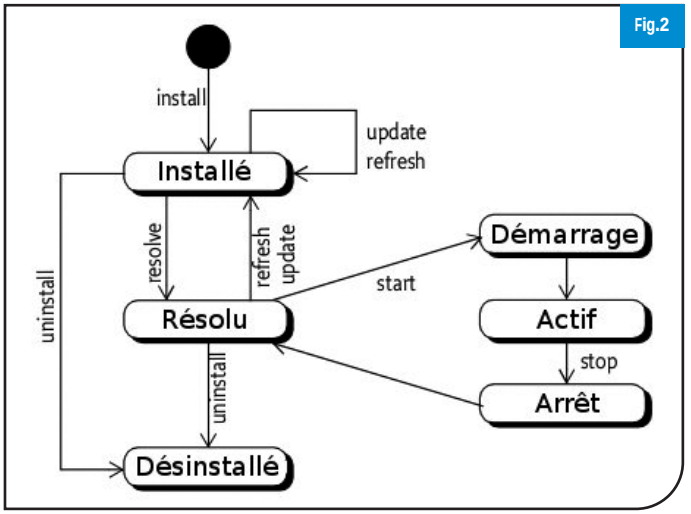
Le déploiement d'un composant *OSGi* se réalise donc simplement en se fondant sur son fichier *JAR* associé, le descripteur de déploiement (le *MANIFEST.MF* présent dans le répertoire *META-INF*) étant lu afin de configurer le composant.

Une fois déployé, le composant passe par différents états avant d'être réellement utilisable. Les conteneurs offrent des facilités dans leurs outils d'administration afin de gérer cet état.

Les différents états par lesquels un composant peut passer sont les suivants :

- ***Installé :*** le composant est simplement présent dans le conteneur;
- ***Résolu :*** les dépendances du composant ont été résolues et ce dernier peut être démarré afin d'être utilisé;
- ***Désinstallé :*** le composant n'est plus présent dans le conteneur;
- ***Actif :*** le composant est démarré et peut être utilisé dans le conteneur.

La figure 2 récapitule l'enchaînement possibles des états les uns par rapport aux autres afin de rendre actif un composant installé et inversement.



EN RÉSUMÉ

Dans cet article, nous avons décrit les fondations de la technologie *OSGi*, technologie visant à mettre en œuvre des programmations très légères par composant et orientée services utilisables avec *Java*. Une des particularités d'*OSGi* consiste en sa gestion fine des dépendances au niveau des *packages* et des services tout en offrant un cloisonnement des *classloaders* des composants. Ainsi, des bibliothèques hétérogènes, voire incompatibles peuvent être utilisées dans différents composants sans souci. Cet aspect peut néanmoins avoir des effets de bord désagréables avec les chargements dynamiques de classes.

Au niveau de l'exécution des composants, la technologie nécessite un conteneur *OSGi*, conteneur utilisant simplement des fichiers *Jar* pour le déploiement tout en se fondant sur les données de paramétrages localisées dans leur fichier *MANIFEST.MF*.

Enfin, précisons que *OSGi* est une technologie montante dans la communauté *Java*. En effet, après avoir été popularisé par l'environnement de développement *Eclipse, OSGi* est de plus en plus utilisé dans les applications *Java,* notamment dans celles hébergées par les serveurs d'applications *JavaEE*.

■ Thierry **Templier** (templth@yahoo.fr)
Architecte Java / JEE – Argia-Engineering
Co-auteur des ouvrages Spring par la pratique *et* JavaScript pour le Web 2.0 *parus aux éditions Eyrolles.*

Programmer OpenGL avec le langage Ruby

2e partie

Programmer OpenGL avec un langage de script, c'est, bien souvent, gagner en temps de développement par rapport à C. Mais travailler avec le langage Ruby présente aussi des particularités dont nous continuons la découverte dans cette deuxième et dernière partie.

Au départ, pour des considérations évidentes de performances, OpenGL est une librairie conçue pour être programmée en C ou C++, ou éventuellement un autre langage natif à pointeurs, comme Pascal. Nous parlons ici de performances à l'exécution. Mais lors de la mise au point d'une scène 3D, la recherche de performance peut se situer dans le temps de développement, et sur ce point précis, on ne peut pas dire que les langages cités plus haut soient particulièrement brillants. La manipulation des types OpenGL, des pointeurs, et le cycle écriture-compilation-édition de liens sont fastidieux et coûteux en temps, ressource finalement plus précieuse que la mémoire vive de nos jours. Même avec la puissance des machines actuelles qui "compilent vite", l'écriture reste difficile. C'est pourquoi nous nous sommes intéressés le mois dernier à la programmation OpenGL avec Ruby. Si celui-ci n'est pas particulièrement un foudre de guerre à l'exécution, la puissance des machines modernes permet cependant d'obtenir des résultats satisfaisants, au moins pour des scènes partielles. Par contre avec Ruby l'écriture du code est très rapide. Mais elle comporte aussi des particularités que nous avons commencé à découvrir le mois dernier. Nous continuons notre découverte aujourd'hui.

1 LES LISTES D'AFFICHAGES OPENGL

La lenteur potentielle d'un langage de script peut dans une certaine mesure, souvent assez large, être compensée par une particularité d'OpenGL qui est la liste d'affichage. Une liste d'affichage est un groupe de commandes OpenGL stockées pour exécution ultérieures. Ces commandes sont exécutées dans l'ordre de stockage dans la liste. Attention, la plupart des commandes OpenGL peuvent faire partie d'un liste d'affichage, mais pas toutes. Une liste d'affichage peut être comparée à une sorte de cache. Implicitement, cela signifie qu'il n'est pas possible de modifier une liste d'affichage après sa création car cela reviendrait à agir sur le cache et donc à perdre le bénéfice de celui-ci. Si les listes d'affichage sont intéressantes dans l'absolu au niveau des performances, elles le sont aussi dans le contexte particulier des environnements X-Window. En effet sous X-Window il est possible de travailler à distance, le client, l'application OpenGL, s'exécutant sur une machine A et le serveur X, le poste qui permet de visualiser le graphisme, s'exécutant sur une machine B. Lorsque l'on travaille avec des commandes OpenGL de façon classique, celles-ci sont autant de requêtes qui voyagent sur le réseau à chaque exécution des fonctions de rendu. Outre l'évident souci de la charge du réseau, la vitesse d'exécution est évidemment pénalisée. En revanche, si l'on emploie une liste d'affichage, celle-ci est stockée sur le poste serveur et exécutée sur le poste serveur. La seule requête qui voyage sur le réseau est celle qui demande le rendu de la liste. On ne se privera donc pas d'utiliser les listes d'affichage en général, et en particulier sur les systèmes Unixes, domaine de prédilection d'OpenGL. Dans la pratique, créer une liste d'affichage est tout ce qu'il y a de simple. Voici un exemple, basé sur l'application GLUT minimale vue le mois dernier et reprenant l'exemple, également du mois dernier, du triangle autour de l'axe des Y et en déplacement le long de l'axe des Z. [Fig.1]. Code complet sur notre site.

Fig.1

```
require 'opengl'
require 'glut'

WIDTH = 300
HEIGHT = 300
X_INIT = 100
Y_INIT = 100

$liste_affichage = 0
$angle = 0.0
$eloignement_maxi = -49.0
$eloignement_mini = -12.0
$eloignement = $eloignement_mini
$pas = 0.2
$sens  = true


display = proc {
  GL.Clear(GL::COLOR_BUFFER_BIT)

 GL.LoadIdentity()
 GL.Translate(0.0, 0.0, $eloignement)
 GL.Rotate($angle, 0.0, 1.0, 0.0)
 GL.CallList($liste_affichage)
 GLUT.SwapBuffers()
```

```
  $angle = $angle + 1.0
 if $angle > 360.0
   $angle = 0.0
  end

 if $sens == true
   $eloignement -= $pas
   if $eloignement < $eloignement_maxi
     $sens = false
    end
 else
   $eloignement += $pas
   if $eloignement > $eloignement_mini
     $sens = true
    end
 end
}



def create_list()
 $liste_affichage = GL.GenLists(1)
 GL.NewList($liste_affichage, GL::COMPILE)
 # ici le contenu de
 # la liste d'affichage
 GL.Begin(GL::TRIANGLES)
   GL.Color3f(1.0, 0.0, 1.0)
   GL.Vertex3f(0.0, 0.5, 0.0)
   GL.Color3f(1.0, 1.0, 0.0);
   GL.Vertex3f(-0.75, -0.5, 0.0)
   GL.Color3f(0.0, 1.0, 1.0)
   GL.Vertex3f(0.75, -0.5, 0.0)
 GL.End()
 GL.EndList()
end

def init()
 GL.ClearColor(0.0, 0.0, 0.0, 0.0)
 GL.ShadeModel(GL::SMOOTH)
 create_list()
end
```

Dans ce code, la liste d'affichage est créée, et c'est au cours de l'initialisation de l'application. Notre fonction *init* invoque la fonction *create_list* dans lequel tout le travail est effectué. Nous savions déjà, depuis le mois dernier que l'affichage *OpenGL* est la garniture d'un sandwich dont les deux tranches de pain sont constituées par les appels aux fonctions *glBegin* et *glEnd* (En Ruby *GL.Begin* et *GL.End* respectivement). Ici on ajoute deux tranches de pain au sandwich avec les fonctions *glNewList* et *glEndList*. Le premier paramètre reçu par *glNewList* est un handle, ou dans la terminologie *OpenGL* un index, disponible pour la création d'une liste. Cet index fait partie d'une plage d'index obtenus via la fonction *glGenLists* dont le fonctionnement ne tombe pas sous le sens, c'est pourquoi nous nous attardons. En C cette fonction est prototypée ainsi :

```
GLuint glGenLists(GLsizei range);
```

Le paramètre reçu *range* est une demande de plage d'index. Par exemple, si *range* vaut trois, *glGenList* va rechercher une plage de trois index disponibles. Si elle trouve cette plage elle va retourner une valeur qui sera le premier index de la plage. Dans notre exemple, cette valeur retournée peut être aussi bien 1 que 4 que 7, ou encore autre chose. Il est important de bien avoir en tête qu'il y a total découplage entre la valeur du paramètre *range* et la valeur retournée. Ainsi si *range* vaut 3 et que la valeur retournée est 1, cela signifie que les index disponibles sont 1,2,3 et que ce sont des valeurs que l'on passera à *glNewList* lors de la création des listes d'affichage. Si la valeur retournée est 7, les index seront 7,8,9. Et ainsi de suite. Il est à la charge du programmeur de garder quelque part l'association entre le premier index disponible et l'étendue de la plage d'index, ce couple d'informations étant notamment nécessaire lorsque vient le moment d'effacer les listes avec la fonction *glDeleteLists*. Rien de plus simple que de provoquer le rendu à l'écran d'une liste d'affichage. Un appel à *glCallList* et le tour est joué.

2 DU BROUILLARD AU MENU

Notre programme d'exemple est sans doute déjà très amusant, mais il manque de réalisme. En effet, malgré les dégradés de couleurs qu'OpenGL sait générer automatiquement entre les sommets de notre triangle, celui-ci conserve un aspect visuel assez dur. Pour remédier à ce problème courant de la 3D, on utilise le brouillard, ou en bon anglais, le *fog* :) Le brouillard apporte un lissage qui augmente le réalisme des objets en en adoucissant les contours. [Fig.2] Il donne aussi l'impression que les objets s'affadissent avec l'éloignement. [Fig.3] Le brouillard, savamment dosé, est indispensable aux applications dans lesquelles il faut donner une impression de visibilité limitée, un simulateur de vol par exemple. Les algorithmes de calculs de brouillard sont complexes et sortent du cadre de cet article. En revanche, la mise en oeuvre avec OpenGL est très simple. Il existe plusieurs algorithmes de brouillard sous OpenGL et nous profitons de l'occasion pour enrichir notre application d'exemple d'un menu contextuel actionné par le bouton droit de la souris. [Fig.4]. Voici le code partiel de l'exemple :

```
require 'opengl'
require 'glut'

WIDTH = 300
HEIGHT = 300
X_INIT = 100
Y_INIT = 100

$fogMode = 0
$fogColor = []
$liste_affichage = 0
$angle = 0.0
$eloignement_maxi = -49.0
$eloignement_mini = -12.0
$eloignement = $eloignement_mini
$pas = 0.2
$sens  = true

def init()
 GL.ClearColor(0.0, 0.0, 0.0, 0.0)
 GL.ShadeModel(GL::SMOOTH)
```

```
  # Création de la liste d'affichage
  create_list()

  # Initialisation du brouillard
  GL.Enable(GL::FOG)
  $fogColor = [0.5, 0.5, 0.5, 1.0]
  $fogMode = GL::EXP
  GL.Fog(GL::FOG_MODE, $fogMode)
  GL.Fog(GL::FOG_COLOR, $fogColor)
  #GL.Fog(GL::FOG_DENSITY, 0.35)
  GL.Fog(GL::FOG_DENSITY, 0.05)
  GL.Hint(GL::FOG_HINT, GL::DONT_CARE)
end

selectFog = proc {|mode|
  case (mode)
    when GL::LINEAR:
      GL.Fog(GL::FOG_START, 1.0)
      GL.Fog(GL::FOG_END, 5.0)
      GL.Fog(GL::FOG_MODE, mode)
      GLUT.PostRedisplay()
    when  GL::EXP2, GL::EXP:
      GL.Fog(GL::FOG_MODE, mode)
      GLUT.PostRedisplay()
    when 0
      exit(0)
  end
}

GLUT.Init()
GLUT.InitDisplayMode(GLUT::RGB | GLUT::DEPTH | GLUT::DOUBLE)
GLUT.InitWindowSize(WIDTH, HEIGHT)
GLUT.InitWindowPosition(X_INIT, Y_INIT)
GLUT.CreateWindow()
init()
GLUT.DisplayFunc(display)
GLUT.KeyboardFunc(keyboard)
GLUT.MouseFunc(mouse)
GLUT.ReshapeFunc(reshape)
GLUT.VisibilityFunc(visible)
# Création du menu pour la gestion du brouillard
GLUT.CreateMenu(selectFog);
GLUT.AddMenuEntry("Fog EXP", GL::EXP);
GLUT.AddMenuEntry("Fog EXP2", GL::EXP2);
GLUT.AddMenuEntry("Fog LINEAR", GL::LINEAR);
GLUT.AddMenuEntry("Quit", 0);
GLUT.AttachMenu(GLUT::RIGHT_BUTTON);
GLUT.MainLoop()
```

3 MANIPULER DES TABLEAUX DE VERTICES

Jusqu'à présent, pour le tracé de notre triangle, nous avons spécifié les sommets un par un, chaque fois en invoquant une fonction *OpenGL*. Cette approche n'est réellement viable que dans certains cas. Il est souvent mieux de travailler avec des tableaux de sommets. Prenons un exemple pour nous fixer les idées et supposons que nous voulions rendre un cube. Celui-ci possède six faces. Si nous travaillons sommet par sommet, nous donnons 4 sommets par face et finalement 24 sommets au total, alors que le cube ne comporte finalement que 8 sommets partagés. Dans un tel cas, il est préférable d'utiliser les routines de tableaux. En interne, *OpenGL* travaille par pointeur sur des tableaux et par déréférencement des pointeurs pour lire les valeurs. Evidemment pointeurs et déréférencement n'existent pas en Ruby, mais il est malgré tout possible et même aisé de présenter les données, à savoir sommets et couleurs associées, sous une forme qui satisfasse les API OpenGL natives. Pour l'exemple nous prenons un polygone à 8 côtés. Voici la partie intéressante extraite de *demo_Array1.rb* :

```
$vertices = 0
$couleurs = 0
$indices = 0

display = proc {
  # etc..
  GL.DrawElements(GL::POLYGON, 8, GL::UNSIGNED_INT, $indices);
  GLUT.SwapBuffers()
  # etc…
  end
}

def prepare_array()
  points =
  [
    [cos(PI/6.0), sin(PI/6.0), 0.0],
```

Fig.2

Fig.3

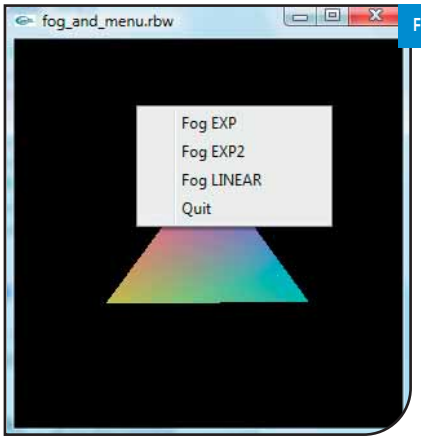


Fig.4

```
    [cos(PI/3.0), sin(PI/3.0), 0.0],
    [-cos(PI/3.0), sin(PI/3.0), 0.0],
    [-cos(PI/6.0), sin(PI/6.0), 0.0],
    [-cos(PI/6.0), -sin(PI/6.0), 0.0],
    [-cos(PI/3.0), -sin(PI/3.0), 0.0],
    [cos(PI/3.0), -sin(PI/3.0), 0.0],
    [cos(PI/6.0), -sin(PI/6.0), 0.0]
  ]

  $vertices = points.flatten()
  $vertices = $vertices.pack("f*")

  $couleurs =
  [
    [1.0, 0.0, 1.0],
    [1.0, 1.0, 0.0],
  ]

  $couleurs += $couleurs
  $couleurs += $couleurs
  # et non pas simplement couleurs.flatten() !! ;-)
  $couleurs = $couleurs.flatten()
  $couleurs = $couleurs.pack("f*")

  $indices = (0..7)
  $indices = $indices.to_a()
  # I pour unsigned int obligatoirement
  # car requis par GL.DrawElements
  $indices = $indices.pack("I*")


  GL.EnableClientState(GL::VERTEX_ARRAY)
  GL.EnableClientState(GL::COLOR_ARRAY)

  GL.VertexPointer(3, GL::FLOAT, 0, $vertices)
  GL.ColorPointer(3, GL::FLOAT, 0, $couleurs)
end
```

La fonction *prepare_array* mérite toute notre attention. Nous commençons par constituer / calculer un tableau de points. Ce tableau est multidimensionnel, car c'est bien pratique pour l'écriture (enfin… question de goût :) Mais *OpenGL* ne travaille pas avec des tableaux multidimensionnels, c'est pourquoi nous demandons à Ruby d'aplatir notre tableau. Unidimensionnel, il devient notre tableau de sommets (ou vertices). Quoique pas tout à fait. En effet, on ne sait pas comment Ruby range les données d'un tableau en mémoire. Mais ce que l'on sait c'est que OpenGL déréférence un pointeur. Donc les données doivent être situées dans une zone continue de mémoire et les données doivent être compactées (c'est-à-dire rapprochées les unes des autres) dans cette zone. C'est pourquoi nous faisons appel à la méthode pack des tableaux Ruby. Cette méthode assure que les données seront rangées en mémoire de façon convenable. Ensuite nous créons un tableau de couleur. Paresseux, nous profitons du fait que Ruby sait additionner des tableaux, puis nous aplatissons et compactons comme précédemment. Ensuite *OpenGL* requiert un tableau d'indices. Nous construisons celui-ci à la Ruby, et toujours nous compactons (attention au type unsigned int). Enfin, nous activons les fonctionnalités de tableaux, pour les vertices et les couleurs avec deux appels consécutifs à *glEnableClientState.* Cette fonction mérite notre attention et d'ailleurs son nom doit nous mettre la puce à l'oreille. Normalement pour activer une fonctionnalité *OpenGL,* on utilise la fonction *glEnable.* Pourquoi pas ici et pourquoi le terme *Client* apparaît-il dans le nom de la fonction ? *Client* signifie "côté client" au sens client / serveur du terme dont nous avons parlé plus haut. En effet, le travail que nous avons effectué avec les tableaux jusqu'ici se passe uniquement côté client. Cela veut dire que les appels à glEnableClientState ne peuvent en aucun cas se situer dans une liste d'affichage, qui est rappelons le, en cache côté serveur. Ensuite, nous spécifions les pointeurs sur les tableaux par les appels à *glVertexPointer* et *glColorPointeur.* Là encore nous sommes côté client, donc pas dans une liste d'affichage. Le troisième paramètre, ici 0, spécifie, l'intervalle éventuel existant en mémoire entre les données. Ruby compactant au maximum, la valeur est toujours zéro. Enfin vient l'affichage, dans la routine *display* comme il se doit, par l'appel à *glDrawElement.* C'est à ce moment que les pointeurs sont déréférencés et les données traitées côté serveur; cela revient à dire que si jusqu'ici nous n'avons pas pu mettre nos appels de fonctions dans une liste d'affichage, l'appel à *glDrawElement* peut, quant à lui, être placé dans une liste; Ainsi le tracé de nombreux polygones via un tableau sera aussi performant et même plus que le même tracé obtenu en spécifiant les sommets un à un. Le lecteur trouvera encore sur site le programme *demo_array_2.rb,* qui exploite le fait qu'au départ, tous les points du polygone se situent dans le plan XY. On ne spécifie donc que des couples de coordonnées, et OpenGL traite le cas correctement, simplement grâce à cet appel:

```
GL.VertexPointer(2, GL::FLOAT, 0, $vertices)
```

L'exemple *demo_array_3.rb* démontre l'utilisation de la fonction *glDrawArray* en lieu et place de *glDrawElements* et l'exemple *demo_array_4.rb* montre comment rendre quelques vertices parmi ceux d'un tableau, au moyen de la fonction *glArrayElement.* Enfin le lecteur trouvera le code *nehe_lesson36.rb.* Ce code, pris sur http://nehe.gamedev.net/ vient avec la distribution *ruby_opengl,* mais sans être mis à niveau pour la version 0.6 de *ruby_opengl* que nous avons *utilisée.* Nous avons donc ré-écrit ce code. Nous pensons que cela pourra peut être rendre service au lecteur, en outre cela montre qu'on peut aller très loin avec Ruby et OpenGL. [Fig.5]

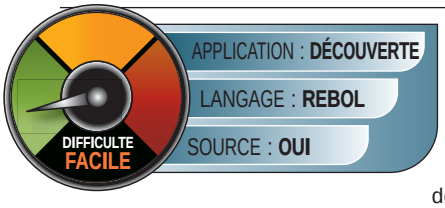
Fig.5

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

A la découverte de REBOL

1re partie

REBOL se prononce rebel. C'est dire si ce langage se veut anticonformiste. Comme tel, il ne s'aborde pas comme un langage de facture plus classique. Mais se familiariser avec lui est amusant et vaut vraiment la peine. Découvrons REBOL ensemble.



REBOL fait partie de ces langages qui tardent à percer, sans doute parce qu'ils sont fondamentalement différents de ceux créés par de grosses compagnies et qui de surcroît bénéficient de leur puissant soutien. REBOL a été créé par un certain Carl Sassenrath. Carl n'est pas le premier venu, car il fut l'architecte du système d'exploitation de l'Amiga. Machine mythique s'il en est, dotée d'un OS non seulement très compact mais aussi multi-tâche 100% préemptif, d'un équivalent de plug-n-play, de deux puces (Blitter et Copper) dédiées au graphisme, d'une puce audio (Paula), etc. C'était en 1985, c'était unique, c'était anticonformiste, c'était en avance sur son temps, et aujourd'hui tous les PC sont comme ça :) La même chose se produira-t-elle avec REBOL ? L'avenir nous le dira. En attendant nous allons essayer de le découvrir un petit peu par la pratique.

1 PRÉSENTATION GÉNÉRALE DU LANGAGE

En 1996 Carl a fait le constat que l'informatique est inutilement compliquée et lourde, et il a créé REBOL, un langage pour le coup extrêmement léger et compact. Le runtime est composé d'un seul exécutable, autonome, sans librairie partagée. La taille de cet exécutable est de l'ordre de 300 Ko en version console et 600 Ko en version graphique [Fig.1]. Un rêve pour le déploiement. Téléchargez la version qu'il vous faut à www.rebol.com, faites pointer cet exécutable par le PATH de votre système et c'est prêt :) REBOL est un langage de script dynamiquement typé, de très haut niveau d'abstraction, très expressif et donc à l'écriture concise. Ses types sont de haut niveau également. Ainsi une URL ou une balise HTML ou encore une adresse IP sont des types en REBOL. D'ailleurs REBOL est conçu pour faire merveille sur le réseau et il intègre en natif le support de SMTP ESMTP POP IMAP HTTP FTP NNTP Finger, Whois, Daytime, TCP, UDP... REBOL est très particulier puisque c'est un langage sans mots-clés. Les mots sont définis dans des dictionnaires et interprétés suivant leur contexte. Il est ainsi possible de redéfinir le mot *if* ou le mot *foreach* si on le souhaite et où on le souhaite :) Si redéfinir un mot comme *if* n'est sans doute pas une bonne idée, REBOL est en revanche tout désigné pour travailler selon un concept à la mode en ce moment, les DSL, ou Domain Specific Language, car REBOL permet d'élaborer des dialectes, c'est-à-dire des langages métier adaptés à une tâche et un contexte précis. REBOL emprunte beaucoup à la programmation fonctionnelle, à LISP notamment et il est parfois qualifié de LISP sans parenthèses. Il emprunte aussi à la programmation objet en y apportant une originalité propre. REBOL n'est toutefois pas une nouvelle panacée. Il a des défauts. En sa version 2.0 (la 3.0 est en préparation) il ne supporte pas Unicode, ni les threads, bien qu'il fonctionne en mode asynchrone dans le travail en réseau. Mais REBOL présente une qualité suprême, très rare : coder avec lui est plaisant et rapide.

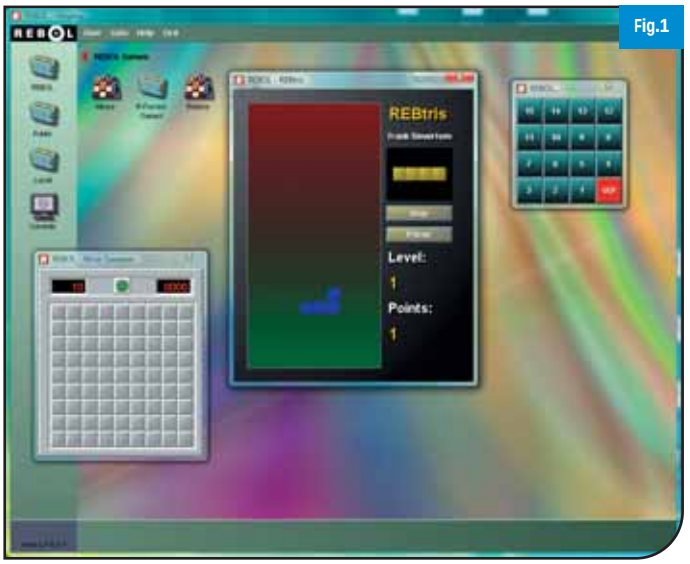


Fig.1

La version graphique de REBOL vient avec un bon nombre de jeux.

2 TRAVAILLER AVEC DES FICHIERS

Mais laissons maintenant la théorie et découvrons quelques aspects de REBOL par la pratique. Un langage qui se respecte se doit d'être efficace en ce qui concerne le travail avec les fichiers texte. Pour voir ce que cela donne, nous supposons avoir deux fichiers conjoints dans un répertoire. Le premier, fruits.txt, contient les trois lignes suivantes :

```
pommes
poires
cerises
```

Le second fichier, baptisé demo-fichier.r et disponible sur le site www.programmez.com, contient le code REBOL. Votre serviteur a écrit ce code avec son inusable Emacs [Fig.2]. Un autre éditeur "qui va bien" avec REBOL est Crimson (www.crimsoneditor.com). Crimson est d'ailleurs recommandé par Carl Sassenrath lui-même. Voici le code REBOL :

```
REBOL [
  Title: "Fichiers 1"
  Version: 1.0
  Author: "Fred"
  Purpose: {Exemple de manipulation de fichiers.}
]

contenu: read %fruits.txt
print contenu
probe contenu

lignes: read/lines %fruits.txt
```

```
print lignes
probe lignes

print pick lignes 2

print "Recherche..."

foreach ligne lignes [
 if find/match/any ligne "p*" [print ligne]
]

print "Recherche i..."

foreach ligne lignes [
 if find/match/any ligne "??i*" [print ligne]
]

print "Recherche et retour de bloc..."

result: make string! 1
foreach ligne lignes [
 if find/match/any ligne "p*" [insert result append ligne " "]
]

print result
foreach r result [print r]
```

On exécutera ce fichier tout simplement avec la commande :

```
rebol demo-fichier.r
```

ou peut-être, selon votre système d'exploitation et votre version de REBOL :

```
rebcore demo-fichier.r
```

Nous invitons le lecteur à se reporter à la documentation en ce qui concerne les différentes options qu'il est possible de passer à l'exécutable REBOL. Tout script REBOL valide doit débuter par un en-tête. Cet en-tête est constitué du mot REBOL et d'un bloc. Un bloc est toujours limité par des crochets [et]. Le contenu du bloc d'en-tête est à la fois conventionnel et optionnel. Un script REBOL valide minimum, ne faisant strictement rien, est donc :

```
REBOL []
```

Ce qui suit l'en-tête est immédiatement exécuté, sauf bien sûr s'il s'agit par exemple de déclarations de fonctions qui seront alors lues et rangées dans le dictionnaire, en attendant d'être invoquées. Avec la première ligne de notre script, nous approchons la puissance du langage

```
contenu: read %fruits.txt
```

Une seule ligne de code, et beaucoup à dire. La fonction read lit le fichier fruits.txt en entier. Le résultat de cette lecture est affecté au mot contenu qui est du coup une variable. C'est le symbole : qui réalise l'affectation. Contrairement à beaucoup de langages, le signe = fait un test d'égalité. Ainsi si après avoir exécuté le script, vous tapez dans l'interpréteur qui est resté ouvert :

```
>> contenu = read %fruits.txt
```

le système répond :

```
== true
```

Maintenant si l'on change la valeur de contenu :

```
contenu: "vide"
```

Le test donne maintenant un résultat faux :

```
>> contenu = read %fruits.txt
== false
```

On apprécie au passage la puissance de fonction de test qui peut être appliquée sur le contenu entier d'un fichier :) En REBOL les noms de fichiers sont des types, et le nom est toujours précédé du signe %. Un piège induit par les habitudes acquises avec d'autres langages serait de donner :

```
contenu: read "fruits.txt"
```

Mais en REBOL cela voudrait dire que l'on donne une chaîne de caractères en argument à *read.* Or la chaîne est un argument que read n'accepte pas. Pour savoir quels sont les types acceptés par une fonction, il suffit de le demander dans l'interpréteur (on devrait plutôt dire évaluateur d'ailleurs) interactif : [Fig.3]

```
>> help read
```

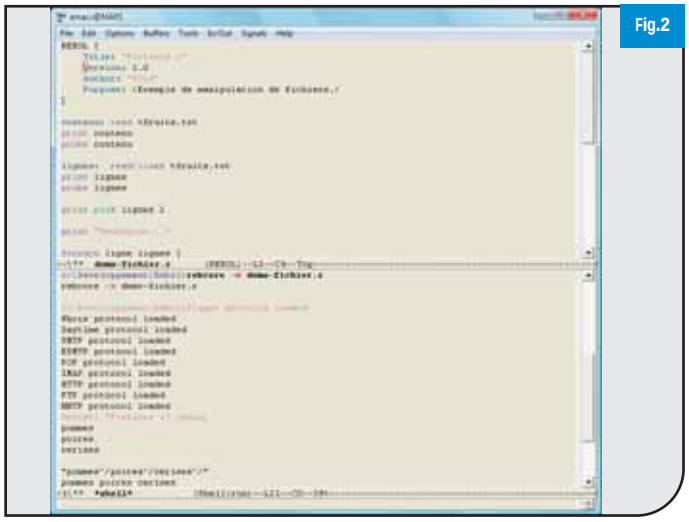


Fig.2

Edition et exécution de l'exemple demo-fichier.r sous Emacs

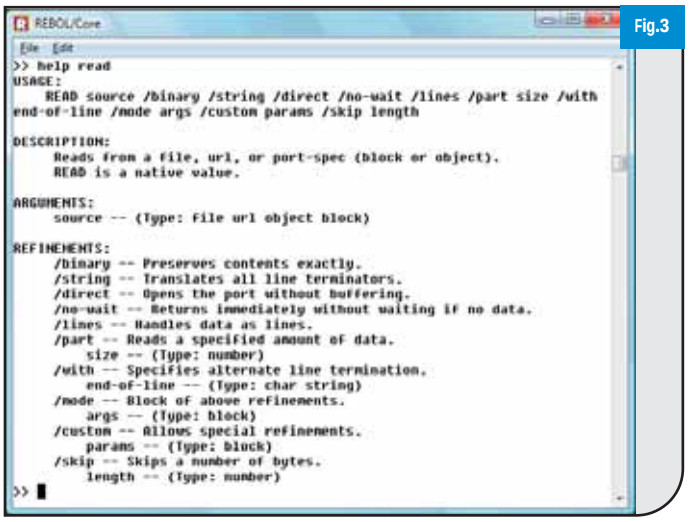


Fig.3

Rebol dispose d'une fonctionnalité d'aide intégrée

```
ARGUMENTS:
  source -- (Type: file url object block)
```

Les types acceptés sont: *fichiers, url, object et block*
On voit donc bien ici que *fichiers* et *url* sont des types natifs à part entière, ce qui, sauf erreur ou omission de votre serviteur est un cas unique dans notre beau monde informatique. Quel que soit le système d'exploitation sous-jacent, c'est toujours le slash (/) qui sert de séparateur, pour les noms de répertoires. Si truits.txt était situé dans un sous-répertoire 'sub' on donnerait simplement :

```
contenu = read %sub/fruits.txt
```

Ensuite, le script imprime le contenu de ... la variable *contenu.* Pour cela la fonction *print* émet une sortie à l'usage de l'humain. La fonction *probe* utilisée immédiatement après émet une sortie non à l'usage de l'humain, mais du programmeur ;) Avec *probe* on peut savoir ce que contient une variable dans le système. La fonction *read* range le contenu du fichier dans une chaîne. C'est pourquoi *probe contenu* nous donne :

```
"pommes^/poires^/cerises^/"
```

soit une chaîne avec des sauts de lignes.

3 LES RAFFINEMENTS DE FONCTIONS

La ligne suivante nous fait vivre un grand moment de programmation car nous découvrons les raffinements de fonction REBOL. Le raffinement d'une fonction est la spécialisation de son comportement. De très nombreuses fonctions disposent de raffinements. Pour les connaître on peut consulter la documentation, ou encore taper *help la_fonction* dans l'évaluateur interactif, comme nous l'avons fait à propos des types. Ainsi nous apprenons que le raffinement *lines* de la fonction *read,* fait que celle-ci manipule le contenu d'un fichier ligne par ligne. Ainsi avec

```
lignes: read/lines %fruits.txt
```

le fichier fruits.txt est ici aussi lu en une seule fois, mais le résultat est rangé dans une liste de lignes. En fait en REBOL, on n'emploie pas le terme de *liste* cher à de très nombreux langages d'inspiration fonctionnelle, mais le terme de *série.* Voila pourquoi *probe lignes* retourne

```
["pommes" "poires" "cerises"]
```

soit une série de chaîne de caractères. Cette *série* est délimitée par des crochets, car une série est avant tout un bloc. Les blocs, et donc les séries, sont l'âme de REBOL. Revenons un instant aux raffinements. Nous avons utilisé le slash (/) pour accéder au raffinement *lines* de *read.* Il est frappant de constater que cette notation est similaire à celle de l'arborescence des fichiers. Mieux que cela, on accède aux propriétés des objets (que nous découvrirons plus tard) de la même manière. C'est encore pareil avec les séries. REBOL offre ici une homogénéité également unique.

4 TRAVAILLER AVEC LES SÉRIES

Voulons-nous récupérer le deuxième élément de la série *lignes ?* Nous utilisons pour cela, parmi d'autres moyens, la fonction *pick.*

```
pick lignes 2
```

Et pour imprimer le résultat :

```
print pick lignes 2
```

Nous arrivons à une autre caractéristique unique de REBOL: sa syntaxe est proche du langage naturel. On code simplement en écrivant les choses de la gauche vers la droite, comme on parle. Avec d'autres langages moins futés, il aurait fallu écrire :

```
print (pick lignes 2)
```

ce qui fonctionne d'ailleurs aussi en REBOL. Mais avec lui, les parenthèses ne sont pas nécessaires. Et comme dit plus haut, REBOL voit les structures de données comme des arborescences, nous pouvons encore écrire :

```
print lignes/2
```

et nous sommes loin d'avoir fait le tour de toutes les possibilités :) La fin de notre premier exemple s'emploie à parcourir des séries et constituer et faire des recherches de chaîne de base. Nous ne détaillerons pas tout. Regardons seulement ceci :

```
foreach ligne lignes [
  if find/match/any ligne "p*" [print ligne]
]
```

Le mot foreach, on s'en doute, permet d'itérer sur une série. L'action qui doit être effectuée à chaque itération, à savoir un test, est placée dans un bloc. L'action qui doit être effectuée si le test est positif est à son tour placée dans un bloc. On remarque à nouveau l'écriture entièrement de gauche à droite. Nous avons aussi l'illustration d'une autre particularité de REBOL: la possibilité d'employer simultanément plusieurs raffinements d'une fonction, ce que nous avons fait ici avec deux raffinements de la fonction *find,* respectivement *match* et *any.*

5 COMMUNIQUER AVEC LE SYSTÈME D'EXPLOITATION

Les versions gratuites de REBOL ne sont pas vraiment conçues pour cela, mais avec un petit peu d'astuce, il est possible de contourner facilement les difficultés. Voici un script (environnement.r sur le site) qui obtient le contenu de la variable PATH du système, le décompose en ses éléments constitutifs, puis affiche ses éléments.

```
REBOL [
  Title: "Environnement"
  Date: 7-Aug-2008
  File: %environnement.r
  Author: "Fred"
  Version: 1.0
]

path: ""
call/output "echo\ %PATH%" path

print path

elements: parse/all path ";"
foreach element elements [print element]
```

Le principe est tout simple. Nous commençons par créer une variable path avec une chaîne vide comme contenu. Cette opération

est nécessaire, car path doit être typé dans la ligne de code suivante. Cette ligne, par la fonction *call,* lance une commande système, ici echo %PATH% (nous sommes sous Windows), et, grâce au raffinement *output* de *call,* ce qui est émis par la commande *echo* est stockée sous forme d'une chaîne dans *path.* Remarque: ce script doit être lancé en désactivant le gestionnaire de sécurité, ceci en passant l'option -s à rebcore.

6 PARSE, LE COUTEAU SUISSE DE REBOL

REBOL dispose d'une fonction d'enfer: *parse*. Un magazine entier de Programmez! ne suffirait pas à en faire le tour. *Parse* est en quelque sorte le couteau suisse de REBOL. Avec elle, il est possible de bâtir des dialectes. Il est aussi, pour nous et plus simplement pour aujourd'hui, possible de décomposer le *path* en ses éléments. Ceux-ci, sous Windows, sont séparés par le signe point-virgule (;). Nous pourrions itérer dans la chaîne *path* pour y rechercher toutes les occurrences de ; et agir en conséquence. Ce serait l'approche classique avec un autre langage. En REBOL, l'instruction *parse* plie le problème en une ligne de code :

```
elements: parse/all path ";"
```

Après quoi, la variable elements contient tous les éléments constitutifs du *path système.* Nous ne saurions trop encourager le lecteur à se familiariser avec l'étonnante et ultra-puissante fonction *parse.*

7 ANALYSER LES ARGUMENTS D'UNE LIGNE DE COMMANDES

Un script REBOL peut recevoir des arguments de la ligne de commande. Voici un exemple de script qui analyse les arguments qu'il reçoit. (fichier args.r sur le site) Ceux-ci sont -h et --verbose=on|off. Voici le code :

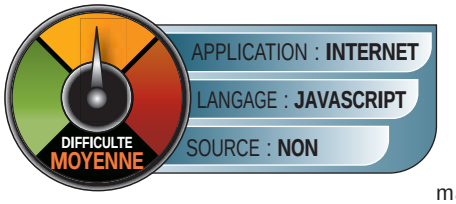
```
REBOL [
  Title: "Args"
  Date: 7-Aug-2008
  File: %args.r
  Author: "Fred"
  Version: 1.0
  Purpose: {Manipulation des arguments de la ligne
  de commande
  options connues: -h --verbose=on|off }
]

verbose: off

get-verbose: func [
  "Traite l'option --verbose dans une chaine d'options "
  all-args [block!] "Les options passées au script"
  /local largs value
][
  foreach element all-args [
    if find element "--verbose=" [
      largs: parse element "="
      either equal? length? largs 2 [
        value: pick largs 2
        if not or equal? value "on" equal? value "off" [
          print "Les valeurs de l'option --verbose sont on ou off"
          print "Arret"
          quit
        ]
        verbose: value
      ][
        print "Option verbose mal formee"
      ]
    ] ; fin if
  ]
]

get-help: func [
  "Traite l'option -h dans une chaine d'options "
  all-args [block!] "Les options passées au script"

][
  foreach element all-args [
    if element = "-h" [
      print "Ceci est un texte d'aide: Programmez! Abonnez vous :-)"
    ]
  ]
]

process-options: func [
  "Applique des fonctions de traitement aux options passées au scripts"
  fonctions [block!] "Les fonctions de traitement"
     all-args [block!] "Les options passées au script"
][
  foreach fonction fonctions [
    do fonction all-args
  ]
]

process-options [get-verbose get-help] parse system/script/
args none
prin "verbose: " print verbose
```

Pour écrire ce script nous nous sommes basés sur ce que nous savons déjà de REBOL. En outre, nous avons voulons montrer un de ses traits, hérité de la programmation fonctionnelle : les fonctions sont des valeurs comme les autres. Le point d'entrée du code se situe tout à la fin. Nous appelons une fonction baptisée process-options en lui passant deux séries en arguments. La première série est une série de fonctions d'analyse, définies en début de script; La deuxième série est une série de chaîne, chaque élément étant un élément constitutif de la ligne de commande. La deuxième série est obtenue avec un petit coup de fonction *parse,* encore elle :) La fonction *process-options* applique chaque fonction de la première série sur la deuxième série. L'intérêt de travailler avec des fonctions comme valeur est que si l'on veut ajouter la recherche d'une option, il suffit d'écrire la fonction correspondante et de l'ajouter à la série de fonctions. Cela marchera sans autre modification du code. Nous laissons le lecteur étudier le reste du code pour y découvrir comment on définit une fonction en REBOL, comment on lui attribue une chaîne de documentation, comment on y définit des variables locales, etc. REBOL nous a déjà révélé quelques uns de ses atouts, malgré l'usage très basique que nous en avons fait. Le mois prochain, nous l'utiliserons à un niveau d'abstraction plus élevé et nous en tirerons encore plus de satisfaction :)

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

Réaliser un composant " glisser-déposer " avec GWT

Google Web Toolkit ne dispose pas en standard d'un mécanisme de " glisser-déposer ". Il existe, certes, des modules développés ici et là qui proposent un mécanisme de ce type, basé sur une bibliothèque Javascript populaire (JQuery, Dojo) ou n'utilisant que l'API de GWT. L'usage d'un mécanisme de ce type pouvant être très varié : déplacer des portlets sur un portail personnalisé, remplir un caddie sur un site marchand, jouer en ligne à un jeu de plateau, une certaine forme de spécialisation est souvent nécessaire. Le " glisser-déposer " est multiple.



En fait, mettre en œuvre le " glisser-déposer " avec GWT est relativement facile, même en partant de rien. La facilité de programmation offerte par GWT, les possibilités de CSS et les performances des navigateurs actuels, nous permettent de faire " le " mécanisme de " glisser-déposer " dont nous avons besoin. Pour un investissement modeste. Un peu de savoir-faire, quelques bonnes pratiques et le tour est joué.

ARCHITECTURE DE NOTRE COMPOSANT " GLISSER-DÉPOSER "

Notre composant sera simple. Il est constitué d'un fond de panier sur lequel l'utilisateur déplacera des images. La place dans cet article étant limitée, nous nous contenterons de gérer les aspects essentiels du mécanisme :

- Création du composant
- Placement des éléments déplaçables
- Réception et traitement des événements souris

Le composant doit pouvoir recevoir des composants enfants (les images) et permettre de les placer très librement. Il existe déjà un composant GWT qui assure ces fonctions : " AbsolutePanel ". Les composants images seront des instances de la classe " Image ". Mais, ce n'est pas tout. Il nous faut en effet prendre en compte dans notre architecture, deux aspects essentiels :

- Capturer les événements souris
- Faire en sorte que les images, au cours de leur déplacement, apparaissent au dessus des autres.

Pour bien comprendre l'explication qui va suivre, rappelons que chaque événement souris est lié à un composant graphique de l'arbre DOM qui représente la page affichée. Ce composant est celui qui apparaît juste sous le curseur de la souris. Dans notre cas, il s'agit soit d'une image, soit du fond de panier (si la souris est placée ailleurs que sur une image). Une solution serait de traquer tous les événements sur l'ensemble des composants impliqués (fond de panier et images (surtout)). Cela revient à multiplier l'enregistrement de " listeners " auprès des composants. Enregistrements qu'il faudra savoir retirer si un composant image est enlevé du fond de panier. Autre point : chaque composant est muni d'un " z-order ", c'est-à-dire un niveau de hauteur. Deux composants ne peuvent partager le même niveau. Une image dont le " z-order " est inférieur à celui d'une autre apparaîtra toujours sous cette image. Il faut donc faire attention pour que lors de son déplacement une image ne vienne pas se cacher sous une autre et disparaître, au moins partiellement. Le niveau " z-order " d'un composant est relatif à son composant parent. Ainsi, si le panel p1 est placé sur le panel p2, tout composant possédé par p1 sera placé " devant " p2 ou n'importe quel composant que p2 possède. Il existe une technique simple pour pallier ces deux problèmes : il suffit de recouvrir notre composant " fond de panier " (avec les images qu'il possède), par un composant " glace " (invisible donc). Ce composant " glace " étant placé au dessus du composant " fond de panier ", c'est lui qui recevra les événements souris. C'est donc uniquement sur lui qu'il faudra enregistrer le listener d'événement souris. Et cet enregistrement est permanent, car la glace n'est jamais retirée (contrairement aux images). Ce composant glace résout aussi notre problème de visibilité des composants au cours de leur déplacement. Il suffit de retirer une image du fond de panier, au moment ou le " glisser " commence et de déposer cette image sur la glace. La glace étant placée au dessus du fond de panier (et donc de toutes les images qu'il contient), l'image déplacée restera toujours visible. En fait, notre composant doit posséder deux glaces qui se superposent : la première sert de zone de déplacement pour les images, l'autre, placée au dessus, sera utilisée pour réceptionner les événements clavier. Si nous n'utilisions qu'une glace, la gestion des événements serait plus ardue et plus aléatoire : en effet, au cours de son déplacement, l'image sélectionnée recevra elle-aussi des événements (car le pointeur de souris sera placée sur elle).

1 CONSTRUIRE LE COMPOSANT

Notre composant n'est constitué que d'une seule classe nommée Board (" tableau "). C'est sur ce tableau, donc, que nous ferons glisser d'autres widgets. Les widgets que l'on pourra déposer sur notre tableau pourraient être de type quelconque.

```
public class Board extends AbsolutePanel {
  int width;
  int height;

  public Board(int width, int height) {
    super();
    this.setWidth(width+"px");
    this.setHeight(height+"px");
    this.width = width;
```

```
    this.height = height;
    buildContent(width, height);
  }
…
}
```

Pour nous faciliter la tâche, nous conservons la taille souhaitée du tableau qui doit être définie à la construction. Notez que cette taille est affectée immédiatement au tableau. Elle doit être fournie sous une forme admissible par CSS. Je me permets de signaler qu'aucun espace ne doit séparer le nombre de l'unité (ici " px "), sinon Firefox refuse de lui donner la taille demandée (à l'inverse d'IE et de Google Chrome). L'essentiel de la construction du composant est bien sûr contenue dans la méthode " buildContent " :

```
AbsolutePanel content;
AbsolutePanel dragGlass;
AbsolutePanel eventGlass;

void buildContent(int width, int height) {
  content = new AbsolutePanel();
  super.add(content, 0, 0);
  content.setWidth(width+"px");
  content.setHeight(height+"px");

  dragGlass = new AbsolutePanel();
  super.add(dragGlass, 0, 0);
  dragGlass.setWidth(width+"px");
  dragGlass.setHeight(height+"px");
  eventGlass = new AbsolutePanel() {
    public void onBrowserEvent(Event e) {
      onSelectBrowserEvent(e);
    }
  };
  super.add(eventGlass, 0, 0);
  eventGlass.setWidth(width+"px");
  eventGlass.setHeight(height+"px");

  DOM.setStyleAttribute(eventGlass.getElement(),
              "background", "black");
  DOM.setStyleAttribute(eventGlass.getElement(),
              "filter", "alpha(opacity=0)");
  DOM.setStyleAttribute(eventGlass.getElement(),
              "opacity", "0");

  eventGlass.sinkEvents(Event.ONCLICK|Event.MOUSEEVENTS);
}
```

Cette méthode construit les trois panels qui seront intégrés au tableau :

- " content " est le panel qui doit contenir les widgets déposés sur notre tableau
- " glassDrag " est le panel sur lequel on fera glisser les widgets pendant les opérations " glisser-déposer "
- " eventGlass " est le panel qui recouvre le tableau afin de happer tous les événements souris.

Un " AbsolutePanel ", par défaut, n'a pas de couleur de fond. Il est donc invisible. Cela convient parfaitement pour " content " et " dragGlass ". Mais pas pour " eventGlass ". En effet, les navigateurs, considérant que le panel ne peut être vu, ils l'ignorent dans le traitement des événements souris ! Il existe une solution assez étrange pour qu'un panel reste invisible tout en recevant des événements souris : on lui affecte une couleur de fond (n'importe laquelle) et on le rend complètement translucide ! Notez que la gestion de l'opacité diverge entre IE et les autres types de navigateurs. Il ne reste plus qu'à demander à GWT d'accepter les événements souris provenant de " eventGlass " à l'aide de l'instruction " sinkEvent ". Ces événements seront réceptionnés par la méthode " onBrowserEvent " de l' " eventGlass ", redéfinie ici pour qu'elle appelle la méthode " onSelectBrowerEvent " du tableau. Terminons la construction du tableau en redéfinissant les méthodes " add " afin que les widgets que l'on veut intégrer dans le tableau soient en fait déposés sur le panel " content " :

```
List widgets = new ArrayList();

public void addW(Widget w) {
  add(w, 0, 0);
}

public void add(Widget w, int x, int y) {
  widgets.add(w);
  content.add(w, x, y);
}
```

La liste des widgets inclus est conservée afin que nous disposions d'un ordre d'affichage équivalent au z-order du navigateur.

2 IMPLÉMENTATION DU " GLISSER-DÉPOSER "

La méthode " onSelectBrower " est de facture très classique : de l'objet Event, on retire la position de la souris au moment ou l'événement est émis. Selon le type de l'événement (clic sur le bouton de la souris, déplacement de la souris, relâche du bouton), une méthode différente est appelée.

```
public void onSelectBrowserEvent(Event event) {
  int x = DOM.eventGetClientX(event)+
          Window.getScrollLeft()-content.getAbsoluteLeft();
  int y = DOM.eventGetClientY(event)+
           Window.getScrollTop()-content.getAbsoluteTop();
  switch (DOM.eventGetType(event)) {
  case Event.ONMOUSEDOWN:
    onCounterMouseDown(x, y);
    break;
  case Event.ONMOUSEMOVE:
    onCounterMouseMove(x, y);
    break;
  case Event.ONMOUSEUP:
    onCounterMouseUp(x, y);
    break;
  }
}
```

Le seul point notable concerne la correction de la position de la souris afin que la coordonnée résultante soit relative au côté haut-gauche du composant tableau : il faut tenir compte, non seulement de la position du tableau, mais aussi d'une éventuelle utilisation des ascenseurs de

la fenêtre du navigateur ! La méthode " onCounterMouseDown " sélectionne un éventuel widget à déplacer. Si la souris est positionnée sur un widget au moment ou le bouton est enfoncé, une référence vers lui est conservé dans l'attribut " dragWidget ". Les attributs " sx " et " sy " conservent la position de la souris relativement au widget sélectionné (afin que l'on puisse correctement repositionner ce widget après un déplacement de la souris)

```
int sx = -1;
int sy = -1;
Widget dragWidget;

public void onCounterMouseDown(int mx, int my,
    boolean shift, boolean ctrl, boolean alt, int mouse)
{
  dragWidget = getDragWidget(mx, my);
  if (dragWidget!=null) {
    sx = mx-dragWidget.getAbsoluteLeft()+
                  content.getAbsoluteLeft();
    sy = my-dragWidget.getAbsoluteTop()+
                  content.getAbsoluteTop();
  }
}

public void onCounterMouseMove(int mx, int my,
    boolean shift, boolean ctrl, boolean alt, int mouse)
{
  if (dragWidget!=null)
    moveWidget(dragWidget, mx, my);
}

public void onCounterMouseUp(int mx, int my,
    boolean shift, boolean ctrl, boolean alt, int mouse)
{
  if (dragWidget!=null) {
    dropWidget(dragWidget, mx, my);
    dragWidget = null;
  }
}
```

Voici le code de la méthode " getDragWidget " qui cherche le widget positionné sous la souris au moment ou le bouton est enfoncé et dont le z-order est le plus haut :

```
int getLeft(Widget w) {
       return w.getAbsoluteLeft()-content.getAbsoluteLeft();
}

int getRight(Widget w) {
  return getLeft(w)+w.getOffsetWidth();
}

int getTop(Widget w) {
  return w.getAbsoluteTop()-content.getAbsoluteTop();
}

int getBottom(Widget w) {
  return getTop(w)+w.getOffsetHeight();
}

Widget getDragWidget(int x, int y) {
  for (int i=widgets.size()-1; i>=0; i--) {
    Widget w = (Widget)widgets.get(i);
    if ((getLeft(w)<=x && getRight(w)>=x) &&
      (getTop(w)<=y && getBottom(w)>=y))
      return w;
  }
  return null;
}
```

La fonction " glisser " est assurée par la méthode " moveWidget ". Notez que c'est elle – lors du premier déplacement de la souris – qui retire le widget sélectionné du panel " content " et qui le dépose sur le panel " dragWidget ". Dans ce cas, c'est la méthode " add " de " dragWidget " qui est appelée. Si le widget est déjà présent sur " dragWidget ", la méthode " setWidgetPosition " est préférée à " add " : sur IE au moins, elle est considérablement plus performante !

```
void moveWidget(Widget widget, int mx, int my) {
  if (widget!=null) {
    if (widget.getParent()==dragGlass)
      dragGlass.setWidgetPosition(
                          dragWidget, mx-sx, my-sy);
    else {
      if (widget.getParent()!=null)
        widget.removeFromParent();
      dragGlass.add(dragWidget, mx-sx, my-sy);
    }
  }
}
```

La fonction " déposer " est assurée par la méthode " dropWidget ". Cette méthode met le widget déposé au sommet de la pile des widgets du panel " content ".

3 EXEMPLE DE MISE EN OEUVRE

Il reste à écrire un petit programme qui construit le tableau et dépose sur lui quelques images qui nous permettront de nous amuser un peu :

```
public class GameViewer implements EntryPoint {

  public void onModuleLoad() {
    Board board = new Board(500, 500);
    RootPanel.get().add(board, 50, 50);
    board.add(new Image("images/romeo.png"), 40, 40);
    board.add(new Image("images/juliette.png"), 200, 40);
  }
}
```

■ Henri **Darmet**
Directeur Technique - Objet Direct / Homsys Group
Objet Direct, filiale à 100% de Homsys Group est une société de conseil, de services et de formation, spécialisée sur les technologies objet et Web. Conseil en méthodologie, en architecture et en urbanisation du SI, développement applicatif, édition et distribution de logiciels.
www.objetdirect.com

livre du mois

Accessibilité des sites web

*Difficulté : ** - Editeur : Eni éditions*
Auteur : Luc Van Lancker - Prix : 45 €

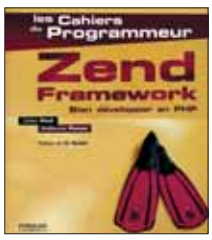
Il existe encore un problème épineux sur le web : l'accessibilité des sites. C'est-à-dire la possibilité de voir, de lire un site pour les personnes ayant un handicap. Dans cette optique, le W3C définit les spécifications WCAG qui permettent de mettre en place les règles d'accès sur les pages web et les contenus. Malheureusement, trop souvent encore, les sites web oublient ce travail ou le font partiellement ou mal. C'est un véritable enjeu informatique autant que social. L'auteur définit tout d'abord l'accessibilité du web et les handicaps puis aborde les outils disponibles tels que les zooms, les loups, les technologies vocales, etc. Le chapitre 5 est particulièrement intéressant car il définit ce qu'est un site accessible. Puis on aborde l'ensemble des éléments en cause : le côté langage, le contenu, les images, le multimédia, les liens, etc. L'ouvrage est particulièrement intéressant. Il devrait faire partie de toute bonne bibliothèque d'un développeur web, d'un webmestre, voire d'un développeur tout court, car l'accessibilité n'est pas uniquement liée au web ! Seul bémol : un prix un peu élevé.

Vulnérabilité et sécurité : le guide des experts

*Difficulté : ****
Editeur : Micro Application
Auteur : Alexandre Gomez Urbina
Prix : 27,50 €

De nombreuses techniques permettent de pénétrer dans un réseau, une base de données, des applications, un système. La généralisation, par exemple, des réseaux wi-fi constitue une source non négligeable des risques. Une des attaques les plus vicieuses est sans doute le keylogger qui capture les saisies claviers ! Nous retrouvons aussi les classiques troyens, virus, etc. Un ouvrage qui permet de se rafraîchir la mémoire et de mettre en place des parades !

Zend Framework



*Difficulté : ****
Editeur : Eyrolles
Auteur : J. Pauli & G. Ponçon
Prix : 39 €

On parle souvent du Framework PHP provenant de Zend. Mais le connaît-on réellement ? Les auteurs, deux spécialistes reconnus de la plate-forme PHP, vous proposent une plongée technique dans cette librairie. Tout y passe. On débute, bien entendu, par l'installation et les éléments de base à connaître (configuration, log, debug, etc.). On entame le développement avec l'accès aux données. Deux chapitres sont exclusivement consacrés à l'architecture MVC du Framework. La sécurité n'est pas oubliée avec deux chapitres : un sur les sessions / authentifications et un autre sur la sécurité proprement dite. On notera aussi une présentation approfondie des performances, de l'internationalisation, des services web, des bonnes pratiques à acquérir, les pratiques de tests. Bref, un ouvrage complet, précis et qui saura trouver sa place auprès des développeurs PHP !

Visual C# 2008

*Difficulté : ****
Editeur : Dunod (Microsoft Press)
Auteur : John Sharp
Prix : N.C.

Visual C# 2008 recèle de nombreuses nouveautés, des astuces de développement et une puissance sous le capot permettant aux développeurs de produire des applications de qualité et toujours plus rapidement. L'auteur revient sur le langage C#, ses principes de fonctionnement, sa syntaxe et les réflexes de base à acquérir dans la programmation C# (erreurs, commandes, opérateurs). Puis on entame la compréhension même du langage avec les classes, les valeurs, les tableaux, les interfaces, les classes abstraites, le garbage collector, etc. Pour être un peu plus pratique, la 4e partie se focalise sur le développement WPF et la partie suivante sur les bases de données. L'ouvrage est plutôt complet et permet de bien démarrer en C# relativement rapidement. Original : la partie développement web est disponible en téléchargement sur le site de l'éditeur !

Et aussi...

XHTML et CSS 2 (collection Le Guide survie, Pearson, 16 €) : vous êtes dans les commandes CSS et XHTML ? Ce guide pratique vous propose de les retrouver rapidement ! Les commandes et fonctions sont regroupées par catégorie avec à chaque fois des exemples de code, des conseils.

Dotclear 2 (Eyrolles, 14,90 €) : comment créer et déployer rapidement son blog ? Dotclear est un puissant gestionnaire de blog à la portée de tout un chacun. L'auteur se propose d'approfondir, via Dotclear, de l'installation à la mise en production de son blog. C'est clair et plutôt didactique.

Algorithmique (collection Les TP informatiques, Eni éditions, 27 €) : l'algorithmique reste une des bases de la programmation. Et la maîtrise d'un langage passe par cette " épreuve ". Ces TP vous proposent d'approfondir avec plus de 140 QCM et exercices les algorithmes et la programmation Java !

Pourquoi peindre avec les doigts ?

Visualisation Java pour clients riches et Ajax

ILOG JViews 8.1, la dernière version de la suite d'outils graphiques Java d'ILOG, couvre l'ensemble des fonctionnalités de visualisation avancée.

ILOG JViews 8.1 offre :

- Des composants graphiques puissants : diagrammes, courbes, tableaux de bord, cartographie, diagrammes de Gantt
- Des services évolués : agencement automatique de graphes, affichage performant pour des jeux de données volumineux
- Plusieurs techniques de déploiement : clients riches, applications Web interactives Ajax, Eclipse/RCP et portails
- Une expertise prouvée dans les industries les plus exigeantes : Informatique, télécoms, transport, énergie et défense.

Testez un de nos produits Java dès aujourd'hui http://jviews.ilog.com

ILOG
Changing the rules of business™