



Programmez!

Ce mois-ci : VOICE XML • C++ • VISUAL BASIC • JAVA • OPENGL • RUBY • PYTHON

- ANIMATION INTERNET
- SÉCURITÉ DES DONNÉES
- DÉVELOPPEMENT ORIENTÉ OBJETS

DÉCOUVREZ LA POLYVALENCE DE

JAVA

Sur le CD-Rom p.35

- Rational Rose 2001
- Outils Microsoft Universal Description Discovery and Integration
- .Net Mobile
- Dreamweaver 4.0

BELGIQUE 260 FB - SUISSE 12 FS
LUXEMBOURG 260 F LUX - CANADA 8,95 \$ CAN

T 4319 - 29 - 39,00 F



Linux:

- ✓ VERSION 2.4 : Les nouveautés
- ✓ À l'intérieur du noyau
- ✓ Les bibliothèques SDL

Pratique

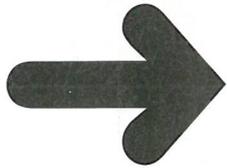
Messaging

LA COMMUNICATION ENTRE APPLICATIONS

- Les pièges du C++
- Le mariage Python/C++
- OpenGL : dernière partie
- Les collections VB.Net

Technologies

- Le langage Ruby
- Les algorithmes RC5 et MD5
- Voice XML



Édito

Numéro 29 • Février 2001

Liberté, liberté chérie



Stéphane Larcher
stephane.larcher@sepcom.fr

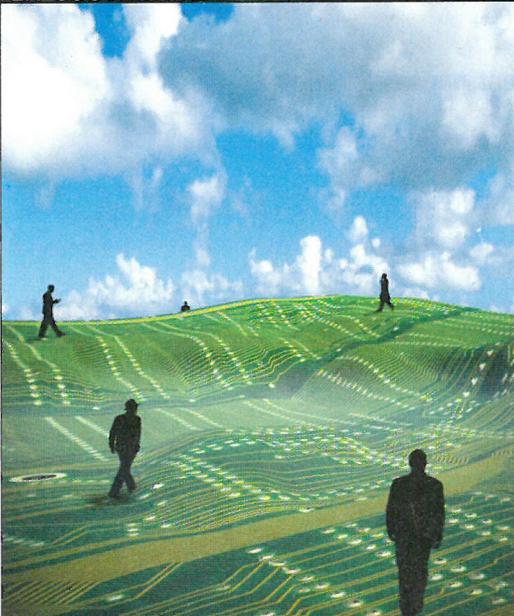
Le débat du mois dernier a suscité de nombreuses réactions pour la plupart positives. Le reproche principal nous est venu de l'APRIL qui a souligné – à raison – une er-

reur concernant le logiciel libre à ne pas confondre avec le logiciel gratuit. Si la version corrigée a été publiée *on line*, nous n'avons pas eu le temps de le faire sur la version imprimée. Au lieu de *Free signifie gratuit*, il fallait lire : « Dans l'esprit de Richard Stallman et de la Free Software Foundation, *free signifie libre dans sa plus totale acception, c'est-à-dire que la fondation est très hostile à l'introduction de logiciels propriétaires sur des systèmes libres, une attitude pourtant conforme aux termes de la GPL* ». Pourtant, et pour en avoir discuté longuement avec plusieurs personnes, c'est encore incomplet, voire inexact.

C'est pourquoi je pense qu'il faut prolonger ce débat et que de nouveaux interlocuteurs prennent la parole et défendent leurs positions. J'ai eu l'occasion de m'entretenir avec le président de l'APRIL et nous sommes convenus d'étendre cette discussion pour, – entre autres choses – préciser les différentes licences qui existent autour du logiciel libre. Mais que les adeptes du tout propriétaire, que les prétendants du « brevet pour tout » se manifestent : leurs arguments pourront être entendus.

De manière générale, continuez à nous transmettre vos critiques comme vos encouragements. Comme vous le constaterez dans le courrier des lecteurs, c'est avec vos réactions que nous avançons, même si nous ne pouvons ni ne voulons vous promettre de toujours suivre vos avis ou vos recommandations.

EN COUVERTURE



La polyvalence de Java

> Pour le développement objet, la sécurité, l'animation de sites web par des appliquestes, Java semble incontournable.

Sommaire

Société Européenne de Presse
et de Communication
5-7 rue de l'Amiral-Courbet, 94160 Saint-Mandé - France
Tél. : 01 43 98 22 22 - Télécopie : 01 43 98 72 12

Éditeur délégué :
Stéphane Larcher

Rédaction
Directeur des rédactions : Michel Barreau
Rédacteur en chef : Stéphane Larcher
Secrétaire de rédaction : François Denivet
(francois.denivet@sepcom.fr)
Assistante de rédaction : Apie-Josa Akaffou
(apie.josa@programmez.com)

Ont collaboré à ce numéro...
Christophe Babayou, Richard Clark, Nicolas Dasriaux,
Alexandre Deloy, François Denivet, Laurent Foynard, John
Hall, Xavier Leclercq, Frédéric Mazué, Médéric Morel,
Carole Pitras, Philippe Prados, Jean-Marc Quéré, Maurice
Szmurlo, Zhang Yong

Maquette
Chef de studio : Marc Soría-Piles
Maquettistes : Laurent Pleissingier, Audrey Ravelli, Claude
Marrel

Services en ligne (ECC Multimédia)
Directeur : Stéphane Kaminsky
(sk@ecc-multimedia.net)

Multimédia/CD-Rom
Directeur : Olivier Delacourt (381)
(olivier.delacourt@sepcom.fr)
Contenu logiciels :
René Robelin et Franck Cayrol
(hotline-cdrom@sepcom.fr)
Intégration et développement :
Gilles Mélinat et Philippe Coupez

Publicité
Directeurs de clientèle : Bernard Bibas (43 65)
et Virginie Leconte (88 36)
Chefs de publicité : Arnaud Fourest (43 60)
et Christophe Benoiste (43 61)
Chef de publicité junior: Bertrand Ponsonnet (43 56)
Assistant marketing : Benoît Gagnaire
Assistante de publicité : Christine Gaugry (43 83)

Diffusion/promotion
Directeur de la diffusion : Thierry Cagnion
Responsable abonnements et promotion :
Vanessa Pénélaud (vanessa.penelaud@sepcom.fr)
Assistante diffusion : Aurélie Denes (43 10)

Sepcom Service Abonnements
22, rue René Boulanger, 75482 Paris Cedex 10
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
abonnements.sepcom@groupe-gli.com

Fabrication
Daniel Nardon

P.C. Net
Directeur général, éditeur: Stéphane Kaminsky
Responsable éditoriale: Marie Bergeonneau
Responsable promotion: Pierre Collette
Webmaster: Benoît Hervier
Technico-commercial: Hervé Daune
Assistante: Armelle Seck

Directeur de la publication,
Président-directeur général
Jean Kaminsky

Groupe Sepcom
SA au capital de 450 000 F

Impression
Imprimerie RotoFrance
Numéro de commission paritaire :
0900 K 78366
Dépôt légal : 1^{er} trimestre 2001

Certains articles signalés sont copyright Linux Journal ©
publiés sous contrat de licence

Ce numéro comporte sur la totalité des exemplaires 2
encarts abonnement brochés entre les pages 34 et 35, et les
pages 66 et 67, un encart abonnement jeté et un CD-Rom.

Tarifs d'abonnement à **Programmez !**

Abonnement (magazine seul) : 1 an - 11 numéros
France métropolitaine : 299 FF TTC

CEE et Suisse : 340 FF TTC

Algérie, Maroc, Tunisie : 367 FF TTC

Canada : 422 FF TTC

Autres pays : nous consulter

Groupe
SEPCOM

HOT LINE CD-ROM

par Minitel : 3614 ISICOM
par Internet : cdrom@sepcom.fr
par téléphone : 01 43 98 43 89
du lundi au vendredi

ACTUALITÉS

Linux en pleine maturité	8
Sash, le langage de scripts d'IBM	9
Livres, notre sélection	10
Un noyau attendu	12
Tasca m'a taxé	13

PRATIQUE

Visual Basic – Systèmes embarqués: optimisez!	14
Java – Faites vos applets sur Internet	18
Java – La sécurité Java 2, une première approche	22
Java – Les inner classes vous optimisent la vie!	28
VB.Net – Le grand défilé de collections VB.Net	32
C++ – Les petits « plus » de C++	38
Python – Ne jetez plus l'argent par les fenêtres	42
Messaging – Les concepts du messaging	46
Linux – Une excursion en mémoire	50
Open GL – Initiation à Open GL (fin)	54

TECHNOLOGIE

Internet – Faites-vous entendre grâce à Voice XML	60
SDL – Un cours accéléré en SDL	73
Ruby – Ruby <i>can't fail</i> , il fait tout!	78
Sécurité – Quand il faut montrer patte blanche RSA/MD5	84

TRAJECTOIRE

Emploi	89
Histoire des langages – Basic, élémentaire mon cher Watson	90
Entretien – Jack Iacobucci surfe sur RogueWave	92
Intelligence Artificielle 2e partie – Faites l'Othello	94
Maison – Tout programmer dans une cuisine	96
Mode de vie – Scénario catastrophe	98

Inondations et tremblements

Ce mois-ci, c'était l'inondation. Vous nous avez englouti sous le courrier. Au secours, à Programmez! on coule. Continuez!

François Denivet
francois.denivet@sepcom.fr

Le courrier des lecteurs peut paraître inutile à certains. C'est un fait, je l'ai entendu dire en termes plus concis et plus triviaux. Par le professeur Choron, qui sait de quoi il parle, dans une émission de télévision, une vitrine publicitaire pour artistes du moment. Ce n'est pas l'avis de «**pestifere314**», fidèle lecteur qui commence son courrier ainsi : «*Préparez vous à subir quelques tonnes d'égoцентриté...*». Suivent des éloges, que nous apprécions en colorant nos joues de pudeur et de satisfaction. C'est d'ailleurs une constante dans vos mails, avec vos encouragements à continuer. La rédaction rougit, et c'est bon.

Ce **pestiféré** qui n'aimait pas les photos de l'ancienne formule « [apprécie] également [notre] optique générale de donner sa chance à chaque langage. L'article sur Python me l'a fait complètement découvrir, poursuit-il. [...] La vision globale sur le monde de la programmation est aussi une grande qualité. On note cependant quelques incohérences : si vous recommandez Win/Webdev, vous ne faites aucun article sur le W-langage. L'initiative du débat libre/propriétaire est excellente ainsi que le commentaire, même si Nat est assez peu agressif, comparé au gars de chez Microsoft. »

Nul doute que nous nous pencherons un jour sur le W-langage. Quant à Python, cette « langue » non fourchue semble susciter l'intérêt. Peut-être exagérément chez **Ludovic P.** qui réclame une « rubrique Python ». Dame, une rubrique, c'est beaucoup plus cher ! Pourquoi pas une rubrique Linux ? Tant qu'on y est. Il est vrai que le cordial et trop modeste **Ludovic** avoue « *apprendre difficilement la programmation (Delphi, C, C++, Java...)*. Il y a quelque mois, précise-t-il, *j'ai découvert Python. Avec ce langage, j'en ai plus appris sur la programmation en deux semaines, que je n'avais réussi à le faire en plusieurs mois d'apprentissage avec d'autres langages. Qui plus est, je comprends et intègre maintenant les autres langages sans grosse difficulté.*

« *Python est d'une grande puissance, de par sa conception objet, entre autres, et d'une grande clarté pour les programmeurs avertis comme pour les néophytes.*

« *La communauté Python est vivace est grandissante...* »

Je vous invite d'ailleurs à feuilleter votre magazine du mois pour le retrouver de nouveau.

Autre constante, ce mois-ci, et pas à notre avantage : le fameux source de l'exemple illustrant l'article « 3-tiers d'architecture, 100% de Visual Basic ». « *Où est-il?* », clamez-vous, interloqués de ne pas le trouver ! Il devait figurer sur le site, il manquait. Vous n'avez pas manqué cette défaillance de notre part. Nous ne l'avons pas joint sur le CD-Rom, pour des impératifs de fabrication (du CD), et l'avons oublié sur le site. Vous avez l'œil, nous rattrapons notre erreur, pardonnez-nous. Ce site – PC Net – fait cliqueter bien des claviers, entre autres, celui d'**Yvan** qui, outre des critiques bien compréhensibles nous suggère de dater les articles, ce qui permet d'évaluer en même temps que la technique. Généralement, dans la version papier de *Programmez!* du moins, les références à d'anciens numéros de votre journal sont indiquées dans le rédactionnel. Yvan nous invite également à continuer, à progresser, et à enrichir les informations concernant les logiciels dont on parle dans le site. Sympathique empressement à voir le perfectible se parfaire. Merci.

Johnny Max – est-il un fidèle lecteur? – nous donne lui-aussi des conseils. Techniques. Il joint son numéro de portable, au cas où nous aurions besoin d'un bon développeur. C'est gentil et charitable, nous gardons ses coordonnées, si vous avez besoin... Nous transmettrons. D'ailleurs nous transmettons un e-mail vers lequel un autre **Ludovic (R)** pourra sans doute trouver des explications concernant l'algorithme de chiffrement RC6 présenté dans le *Programmez!* de décembre. C'est celui des auteurs Christophe Babayou et Laurent Foynard (qui récidivent ce mois-ci). On peut les joindre à contact@uplog.com.

Dans l'ensemble, vous suggérez beaucoup et bien. Je cite en vrac et en en oubliant : parler de l'assembleur, de la programmation d'une chaudière avec un automate, de BeOS, du portage en général et aussi en particulier, des VxD et de l'intérieur de Windows (c'est-y bien propre là-dedans?). Nous essaierons de le contenter. Et vous aussi. En espérant que vous continuerez à nous faire savoir vos impressions, que vous exigerez toujours plus de nous.

Le courrier des lecteurs, c'est notre échelle de Richter à nous, mais c'est aussi votre tribune. Vous avez des choses à dire, des points de vue à défendre. Quand vous élevez la voix, ça tremble. On mesure l'amplitude des secousses, et on vous en fait part. Juste retour des choses. ■

Évolution

Linux en pleine maturité

Attendu tout au long de l'année 2000, c'est finalement le 4 janvier 2001 que le nouveau noyau Linux est sorti dans sa première version stable. Le kernel 2.4 apporte une série d'évolutions importantes mais n'est pas « révolutionnaire » comme l'avait été le passage à la série 2.0

Le noyau 2.4 est sans doute une étape critique pour Linux dans sa conquête des entreprises. À l'heure où ce système libre s'impose comme le rival de MS-Windows 2000, l'arrivée de ce noyau doit entériner définitivement sa supériorité technique en termes de souplesse et de performances. Pour commencer, le Kernel 2.4 permet de voir grand, les limites inhérentes à la version 2.4 ont largement été repoussées. Ainsi, Linux est en mesure de gérer jusqu'à 64 Go de RAM (!) sur les serveurs PC x86. Ces derniers pourront gérer simultanément 16 cartes ethernet et 10 contrôleurs IDE sur une même machine, de quoi voir venir... Par ailleurs, le kernel 2.4 exploite sans difficulté jusqu'à 32 processeurs simultanément. De nouvelles architectures ont également été introduites, le fameux Intel Itanium bien sûr, mais aussi les *mainframes* IBM S/390. Voilà pour les serveurs, mais la souplesse de Linux lui permet de prétendre également aux systèmes embarqués : les processeurs superH utilisés dans les pocketPC d'Hitachi ainsi que les Crusoe de Transmeta (société qui emploie Linus Torvalds) sont maintenant accessibles pour les Linuxiens. Avec ses « cousins » de la famille BSD, Linux est sans doute le système le plus ouvert en terme d'architecture.

Du côté des machines de bureau, le support de l'USB et du *Firewire* sont désormais totalement intégrés. Bien sûr, les utilisateurs du noyau 2.2.18 savent déjà utiliser un bon nombre de périphériques USB. Mais cela impliquait systématiquement une recompilation du noyau et de pilotes plus ou moins expérimentaux avec les aléas que cela sous-entend. Désormais, les choses devraient être beaucoup plus simples.

Un réseau entièrement revu

La couche réseau de Linux a été totalement réécrite. Les fonctions de pare-feu du noyau sont désormais gérées par Netfilter, un ensemble de modules permettant un filtrage plus fin des paquets de données. Cette remise à plat de la partie réseau du *kernel* risque de réclamer une réécriture de certains scripts système, comme ce fut parfois le cas avec le passage au noyau 2.2. Toutefois, au vu des coûts qu'une telle opération ne manque pas d'engendrer dans les grandes entreprises, Linus semble avoir conçu un certain nombre de mesures visant à limiter la portée du problème.

Un sous-système de filtrage IP, baptisé « IP Tables », remplace ainsi les fonctions « ipchains » du noyau 2.2. Ce filtrage devrait réduire considérablement les temps de réaction des serveurs à l'arrivée d'une requête. Enfin, un serveur web élémentaire, dénommé « *khttpd* », a été inclus à l'intérieur du noyau. Il peut répondre aux requêtes les plus simples – les pages html statiques – en amont d'un serveur web. Il devrait donc accélérer les temps de réponse des serveurs web en prenant lui-même en charge la diffusion des pages statiques.

Voilà un petit aperçu de ce que l'on peut attendre du nouveau noyau de Linux. Pour l'instant, vous pouvez le télécharger sur www.kernel.org ou l'un de ses nombreux miroirs, mais il est certain que les prochaines distributions de Linux devraient l'intégrer. La première à le faire sera sans doute Red-Hat qui prépare ce projet sous le nom de code de « Florence ». Patientons encore un peu. ●

■ Sun propose un support complet pour StarOffice

Sun vient de mettre en place une structure d'offre de services de support, de formation et d'accompagnement pour StarOffice. Ces services peuvent être disponibles en ligne, à l'adresse

www.sun.com/support/forum/staroffice, mais aussi par support téléphonique payant. Les tarifs proposés pour un support technique varient suivant que le client est un grand compte, une PME ou une institution d'enseignement universitaire.

■ L'ICANN sur la sellette

Déjà critiquée de toute part, L'ICANN (*Internet Corporation for Assigned Names and Numbers*) est cette fois-ci montrée du doigt par une commission parlementaire américaine. Cette dernière s'interroge sur la pertinence des choix de l'organisme régulateur de l'Internet en matière de création de nouveaux suffixes pour adresse web. « *Selon certains rapports, le processus de création d'une nouvelle génération de noms de domaine Internet par l'ICANN serait de nature à nuire à la concurrence dans le domaine de l'enregistrement et de l'attribution des noms de domaine* », déclare le président de la Commission, le Républicain Billy Tauzin. Rappelons que les derniers choix de l'organisme de régulation portaient sur .aero, .coop, .info, .museum, .name, .pro, et .biz. Ces choix avaient été opérés à partir de 44 propositions et avaient soulevé un grand nombre de protestations. D'autant que la légitimité représentative de cette organisme est plus que contestable.

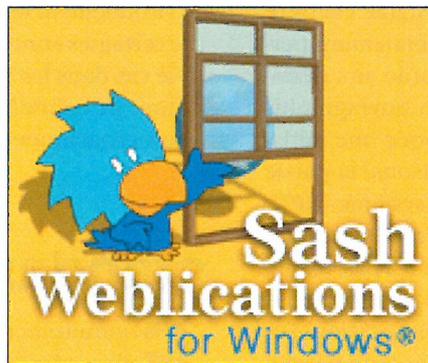
Langage

Sash, le langage de scripts d'IBM

Sash Weblications 2.0 est un langage de scripts créé par IBM et fonctionnant sur les systèmes Windows et Linux. Sash se base principalement sur JavaScript, HTML/DHTML et XML et vise à la construction simplifiée de programmes web. De fait, il est en mesure d'interpréter directement des instructions JavaScript. Le « plus » de Sash est qu'il dispose également d'une batterie d'instructions lui permettant d'accéder à certaines parties du système d'exploitation. Évidemment, plusieurs options de sécurité sont présentes pour limiter les dangers inhérents à ce genre de langages de scripts. L'authentification des applications est possible, et l'utilisateur peut refuser l'activation de certaines fonctions. Les applications écrites en Sash sont exécutées côté client et nécessitent, en plus de l'environnement d'exécution, un navigateur récent (Netscape Navigator ou Internet Explorer à partir de 4.1). L'objectif avoué d'IBM est d'attirer les développeurs web rompus aux langages de

scripts vers leurs plates-formes Websphere et Domino. Pour l'instant, Sash est encore considéré comme un prototype par IBM. Mais il existe déjà quelques applications pour intranet conçues sur cette technologie. Pour conclure, les programmeurs avancés seront ravis de savoir que les sources de Sash sont disponibles en *Open Source* (licence IBM). ●

<http://oss.software.ibm.com/developerworks/opensource/sashxb/>
<http://sash.alphaworks.ibm.com/>



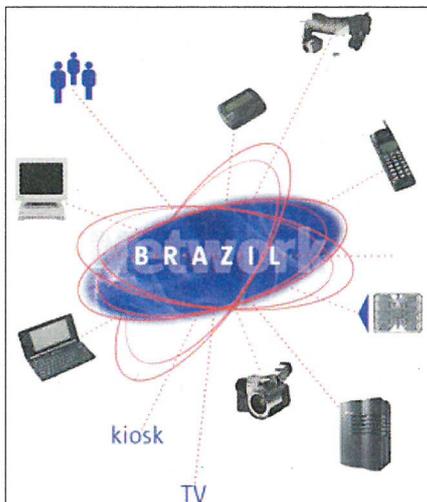
Développement

Sous le soleil du Brésil

Sous le nom de code de Brazil, Sun s'apprête à révéler enfin sa solution de pointe pour le développement d'applications disponible *via* Internet. Une présentation officielle est prévue le 5 février à San Francisco au sujet de ce projet en cours au Sun Labs, aboutissement de deux ans de travaux. À la suite des dernières annonces d'Oracle, Sun annonce donc sa contre-attaque au projet .NET récemment lancé par Microsoft. Brazil ne concerne pas exclusivement les infrastructures informatiques aux sens strict. Nos téléphones portables, cartes à puces et autres objets *high-tech* seront aussi concernés par cette solution globale. Sun a également lancé une nouvelle version de J2EE (Java 2 Platform Enterprise Edition). Le logiciel J2EE étend définitivement la technologie Java sur la scène de l'entreprise. Les apports de cette version 1.3 concernent une simplification de la connectivité. Les autres

nouveautés de J2EE 1.3 sont JMF 2.1 (Java Media Framework), EJB 2.0 (Enterprise Java Beans), et le support XML intégrant JAXP (Java API) et JAXM (Java API pour les messages XML). ●

<http://www.sun.fr/>



Borland au téléphone

Borland et Motorola viennent de s'engager dans un partenariat afin de fournir l'environnement de développement Jbuilder 4 de Borland aux développeurs voulant créer des applications pour la prochaine génération de téléphones sans fil de Motorola... JBuilder 4 Foundation de Borland est un environnement de développement en Java qui supporte les derniers standards de la plate-forme Java 2 et le développement multi-plates-formes.

Cuisine high-tech



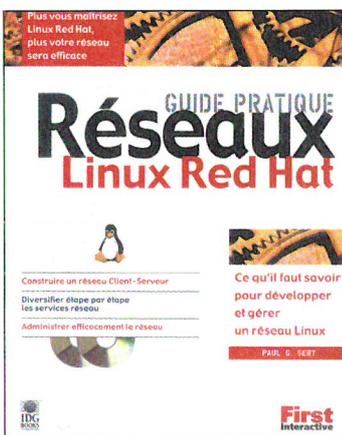
Certains lecteurs ont été chagrinés par notre rubrique « cuisine ». Pourtant, les ustensiles culinaires *high tech*, comme ceux proposés par Digital Cookware ne manquent pas d'attraits. Une cyber-cuisine qui se respecte doit impérativement disposer de sa *Digital Smart Pan*: une poêle intégrant dans son manche un contrôleur de température couplé à un système de programme de temps de cuisson. Vous voilà en mesure d'établir scientifiquement la préparation de votre steak ou de vos œufs au plat. Si vous êtes plus porté sur le plein air et les viandes grillées, il existe également, une « *Digital BBQ Taste-Temp Fork* » qui comme son nom l'indique est une fourchette à barbecue permettant d'évaluer d'un geste si vos côtes d'agneau sont bien à point et votre pièce de bœuf saignante. www.DigitalCookwareinc.com/

Guide pratique Réseaux Linux Red-Hat

Conçu dans une perspective résolument pratique, cet ouvrage s'adresse *a priori* aux personnes ayant déjà acquis des connaissances élémentaires sur Linux et cherchant à s'initier à l'administration de services réseaux. Comme son nom l'indique, le propos de l'auteur concerne la distribution Red-Hat qui d'ailleurs est fournie sur deux CD. On peut, par contre, s'étonner du fait qu'il s'agisse d'une version 6.1. Nous supposons que c'est la version qui fut incluse dans la version originale américaine de l'ouvrage. Une petite mise à jour et ré-adaptation aurait été bienvenue. En dehors de cela, ce livre remplit bien son rôle et permet au lecteur de faire un tour d'horizon résolument pratique sur les services réseaux sur Linux. Bien sûr, on n'échappe pas à l'inévitable chapitre consacré à l'installation de Linux que généralement personne ne lit. Cela peut sans doute être utile aux débutants complets, mais soyons réalistes, un ouvrage plus complet sera de toute façon nécessaire pour une initiation à partir de zéro. Pour les mêmes raisons, la partie consacrée à la configuration générale du système (RPM, Xwindow, etc.) ne présente qu'un intérêt limité. Par contre, la composante physique du réseau n'a pas été négligeable, ce qui est appréciable lorsque l'on ne dispose que de connaissances très théoriques sur la question. Les principaux services sont développés tout au long des différents chapitres : Samba, serveurs DNS, *firewall*... On trouve même quelque pages consacrées à Arkeia (présent sur les CD en version d'évaluation), le fameux système de sauvegarde automatique propriétaire. Quelques absences de taille sont cependant à déplorer. Il n'est nulle part question d'Apache ou d'un logiciel de *mail* performant comme Sendmail ou Qmail. En fait, le livre se concentre sur l'intranet, au détriment des questions de serveurs Internet. Somme toute c'est le parti-pris de l'auteur, qui parle bien de « *construire un réseau Client-Serveur* » en

accroche de la couverture.

En dépit du caractère ancien de la distribution proposée, ce livre présente l'avantage d'être synthétique et résolument pratique. Si vous êtes à la recherche d'une solution réseau, vous y trouvez un mode d'apprentissage pertinent pour environ 250 F. ●

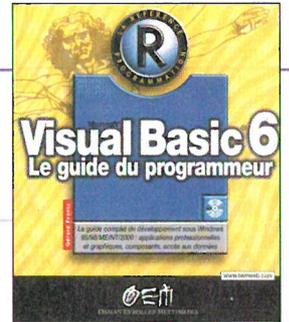


Auteur : Paul Sery pour IDG Books
Édition française : First Interactive

Microsoft Visual Basic 6

Auteur : Gérard Frantz

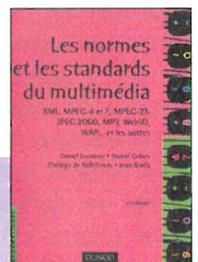
Édition : OEM



Presque 1272 pages, accompagnées d'un CD-ROM contenant tous les fichiers sources exposés. Ce pavé de la collection « la référence programmation » est un guide complet qui ne s'adresse pas forcément aux débutants. Depuis la version 4 qui avait introduit la notion de classe, Visual Basic a élargi son champ d'application. La version 5 supporte la création de composants ActiveX et la version 6 enfin, s'intéresse au développement d'applications réseaux et Internet. Autant de chapitres abordés parmi les 36 présents. On s'y égare un peu d'ailleurs. Un bon point pour les captures d'écran de l'environnement de développement, puisque tout est en français (certaines sont floues aussi!). Ce qui est loin d'être toujours le cas dans les ouvrages de programmation. Ce guide coûte tout de même 328 F, mais est indéniablement complet pour un programmeur ayant déjà des notions de VB. ●

Les normes et les standards du multimédia

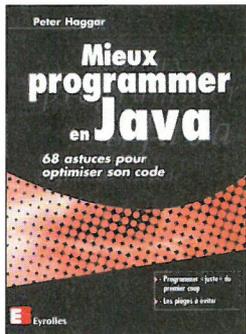
Contrairement aux deux ouvrages précédents, celui-ci est plus orienté sur la théorie. L'idée est bien là, d'appréhender les standards de codage pour la diffusion d'objets multimédias : MP3, JPEG, MPEG-4, etc. L'ouvrage décrit l'état de l'art des moyens permettant de mettre en œuvre les cinq composants du multimédia : la numérisation, le traitement, l'archivage, le transport et la restitution des données. Sans l'approche d'une standardisation, des activités aussi différentes que la production vidéo, l'informatique et les télécommunications ne seraient tout simplement pas en mesure de dialoguer. Les normes permettant l'interopérabilité des équipements, des logiciels, et des infrastructures, assurent la pérennité même du système. La confluence de normes s'est largement accélérée ces dernières années avec l'apparition du réseau Internet, qui exige une accessibilité des données à l'échelle mondiale. Le HTML est d'ailleurs la norme de diffusion d'informations la plus universellement supportée sur les systèmes actuels. Il est également nécessaire de bien peser les valeurs respectives et les capacités que l'on peut obtenir entre différents formats. Les algorithmes que ces derniers exploitent sont également analysés et détaillés. Quel que soit le langage de programmation de prédilection, ce livre s'avère précieux pour peu que l'on ait à manipuler des données multimédia à travers une application ou une base de données. Utile également si l'on travaille à la diffusion ou à la création de telles données. ●



Auteurs : D. Lecompte, D. Cohen,
P. de Bellefonds, J. Barda
Édition : Dunod

Mieux programmer en Java

Auteur : Peter Haggar
Collection Eyrolles



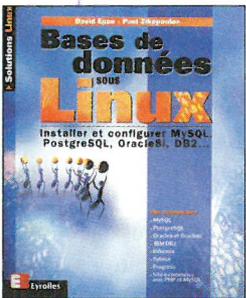
Le débogage représente souvent la part de travail la plus longue pour un développeur. D'où la nécessité de suivre une stratégie de développement rigoureuse afin de limiter au possible les difficultés. C'est précisément le propos de cet ouvrage qui vise à donner, à travers 68 astuces, les moyens de contourner les pièges classiques de la programmation Java. L'objectif est de programmer « juste » du premier coup, et d'optimiser dès le départ les performances du programme. Les points abordés sous la forme d'ateliers traitent différents aspects du langage dans un ordre croissant de difficulté. Les exemples sont nombreux et détaillés afin de préserver le caractère pratique de l'ouvrage. ●

performances du programme. Les points abordés sous la forme d'ateliers traitent différents aspects du langage dans un ordre croissant de difficulté. Les exemples sont nombreux et détaillés afin de préserver le caractère pratique de l'ouvrage. ●

Base de données sous Linux

Auteurs : D. Egan, P. Zikopoulos
Collection Eyrolles

En matière de bases de données, Linux possède deux atouts majeurs, en dehors de sa haute stabilité. D'une part, c'est la plate-forme de référence pour les systèmes de bases de données libres telles que PostgreSQL ou MySQL, particulièrement adaptées pour les petites structures et les sites web dynamiques. D'autre part, les quatre plus grands SGBD commerciaux existent aussi en version Linux. Ce livre aborde l'installation, la mise en œuvre et l'exploitation de tous ces SGBD (MySQL, PostgreSQL, Oracle8i, IBM DB2, Sybase, Infomix etc.). ●

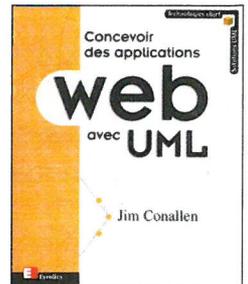


Les responsables des services informatiques et réseaux d'une entreprise sont toujours confrontés au double problème de l'optimisation et de l'évolution de l'infrastructure. Cet ouvrage vise à aider le lecteur à comprendre les enjeux de la qualité des services, afin de mieux en évaluer les coûts et prévoir ensuite une planification des évolutions prévisibles en fonction des besoins de l'entre-

Concevoir des applications web avec UML

Auteur : Jim Conallen
Collection Eyrolles

Ce livre concerne un public déjà initié à la syntaxe et aux principales fonctions de l'UML. Comme tous ceux de la collection « Technologie objet, solutions UML », il vise à donner une méthodologie pour aborder un projet d'envergure. Pour illustrer ce propos, le lecteur peut suivre la création pas à pas d'une application de commerce électronique. Il faut souligner que l'auteur de ces pages est d'abord le créateur de l'extension web du langage UML. Un ouvrage de référence, donc. ●



Applications Cobol sur le web



Le titre de ce livre devrait en laisser perplexe plus d'un. Mais le parti pris de ses auteurs est intéressant : à l'heure de la création d'applications web, faut-il jeter aux orties toutes les applications Cobol existantes pour tout reconstruire sur de nouvelles bases ? Répondre par l'affirmative implique un coût de développement énorme pour les entreprises, alors qu'il existe des méthodes pour faire migrer en douceur les applications Cobol. Bien sûr, si vous n'avez pas de vieilles applications de ce type dans votre parc, ce livre n'a aucun intérêt pour vous. ●

Auteurs : R. Engel, P. Silverio
Collection Eyrolles

Gestion de la qualité de service, réseaux, serveurs et applications

Un bon système d'information est un des éléments clés du succès d'une société quelle qu'elle soit ; vous trouverez ici les méthodologies nécessaires pour une politique d'administration efficace. ●

Auteur : Bruno Fouquet
Collection Eyrolles

Système Linux

Un noyau attendu

Linux 2.4 vient d'être annoncé. De nouvelles fonctionnalités apparaissent : nouvelles architectures, réécriture de certains sous-systèmes, gestion de nouveaux périphériques... Quel est l'intérêt industriel de ces innovations ? Faisons le point.

Prenons l'exemple d'un serveur web Apache, basé sur une machine bi-processeurs avec 2 Go de RAM. Les nouveautés du noyau 2.4 vont permettre de gagner en efficacité et en disponibilité de service. Cette dernière sera accrue par le support des systèmes de fichiers journalisés (XFS, ReiserFS, ext3). Bien qu'à l'heure actuelle, ceux-ci ne sont pas intégrés au sein même du noyau, il est probable qu'ils le seront dans le courant de l'année, en attendant une refonte éventuelle de cette partie dans le prochain Linux, dans le but de séparer la journalisation proprement dite des différents systèmes de fichiers. L'efficacité sera améliorée au niveau de l'agenceur, du module khttpd, et de la couche réseau.

Le nouvel agenceur gère les processus différemment. Dans Linux 2.2, tous les processus en attente d'un événement réseau sont réveillés lorsque ce dernier survient. Or, un seul d'entre eux s'avère être le destinataire réel de cet événement. D'où une perte de performance (changement de contextes d'exécution trop nombreux et inutiles). Linux 2.4 propose une nouvelle méthode « *wake one* » qui établit un lien direct entre un événement réseau et le processus concerné. Lui seul sera réveillé par l'événement, d'où un gain de performance. Apache est un exemple d'applications qui mettront en œuvre cette fonctionnalité.

Encore expérimental à ce jour, le module « khttpd », permet de servir des pages HTML statiques directement à partir du noyau, qui gère par ailleurs les connexions réseau. Le basculement *kernel-space/user-space* n'est donc plus nécessaire dans ce cas, et, là aussi, les performances sont au rendez-vous. Si la demande correspond à des pages dynamiques (script CGI, par exemple), elle est transmise au programme *user-space* concerné (typiquement

Apache). Enfin, la couche réseau de Linux 2.4 a été partiellement réécrite. L'un des objectifs visait un meilleur support SMP (*Symmetric MultiProcessing*). Dans l'exemple que nous évoquons, les performances seront ici aussi accrues. Toujours sur la partie réseau, notons encore la disposition de fonctionnalités de « *bonding* » permettant d'agrèger plusieurs connexions comme une seule dans le but d'augmenter le débit, le support de l'ATM et la possibilité de mettre en place un mandataire transparent (« *frame diverter* »).

Vers l'embarqué

C'est également du côté des capacités du noyau que la version 2.4 a grandement progressé, et ce, par le support de 4 Go de mémoire vive sur architecture Intel, voire 64 Go en utilisant le *Physical Address Extension* (PAE) ; la limite des fichiers à 2 Go est également tombée, même sur des architectures 32 bits, tandis que l'on peut gérer maintenant 4 milliards d'utilisateurs et de groupes. La prise en compte des volumes logiques (LVM) est également intégrée, apportant plus de souplesse dans la gestion et l'utilisation des données, en particulier sur les supports amovibles.

Le spectre des architectures et périphériques supportés s'élargit. Du côté des processeurs, l'ensemble de la gamme compatible Intel est gérée, du i386 jusqu'au prochain Itanium (intégrant la technologie IA64). Corrélativement, la prise en compte des caractéristiques de processeurs spécifiques – comme l'Athlon – et une meilleure gestion du SMP accroît l'efficacité du noyau. Dans le domaine des architectures supportées, citons les S/390 (*mainframe IBM*), les MIPS64 (SGI 64 bits), PA-RISC, et Hitachi Super-H (intéressant pour l'embarqué). La gestion des bus est améliorée, notamment celle des PCI – dont les pilotes

ont été réécrits – de l'EISA, ainsi que de l'AGP, bus rapide utilisé par les cartes vidéo récentes.

Nous pouvons donc observer que Linux investit le monde des systèmes embarqués : la gestion des processeurs dédiés (Super-H, Crusoe,...), ou encore des *Memory Technology Devices* (MTD) pour les cartes flash figurent parmi les apports de ce noyau.

De plus, la prise en compte de l'*Universal Serial Bus* (USB) ouvre d'un seul coup Linux à un ensemble de périphériques facilement installables. D'ailleurs, l'intégration du support *Plug and Play* au sein du noyau conforte cette volonté de simplifier la gestion des ressources, même avec des matériels moins récents (ISA, par exemple). La gestion du PCMCIA, autrefois assurée par un ensemble de modules développés séparément du noyau, est désormais intégrée. On retrouve également la connectique *FireWire*. Tout cela contribue à la cohérence de l'ensemble des fonctions dont doit s'acquitter un noyau de système d'exploitation, tout en garantissant un cycle de vie plus important aux matériels acquis.

Le contrôle de réseau a lui aussi été réécrit : le *masquerading* (autorisant l'accès à l'Internet de plusieurs machines situées derrière une seule IP publique) et le contrôle des paquets sont dorénavant assurés par NetFilter. La gestion de ces fonctionnalités se révèle beaucoup plus modulaire qu'auparavant, tout en préservant la compatibilité avec IPChains (noyau 2.2) ou IPFWADM (noyau 2.0). Une première ébauche de gestion de la qualité de service est intégrée au noyau. On pourra envisager une augmentation de charge avec sérénité et souplesse en veillant à assurer la préservation d'un service équitable, ou en différenciant au contraire les priorités du réseau.

En conclusion, Linux 2.4 apporte une première série de fonctionnalités intéressantes et immédiatement opérationnelles, tout en posant des jalons pour les évolutions à moyen terme. Cette approche conforte une politique de mise à disposition de nouveaux noyaux stables selon une fréquence plus soutenue et favorable, là encore, à l'industrialisation de ce système. ●

Thierry Mallard

Remerciements à Gabriel Forté, Alain Lesné, Nat Makarevitch et Fabien Seisen pour leur relecture.

Taxe

Tasca m'a taxé

L'adage préféré ayant cours au ministère des Finances va à nouveau se vérifier : « *les impôts invisibles sont toujours les meilleurs.* » Catherine Tasca, ministre de la Culture et de la Communication, réaffirme son intention de taxer « *tout support permettant d'enregistrer des œuvres.* ». Pour parler clairement, après les CD (taxe de 4,43 F) et les DVD (29,60 F), Une taxe sur les micro-ordinateurs est maintenant en chantier. Vous étiez déjà ravis de financer votre *boys-band* préféré (et ses producteurs) à chaque sauvegarde gravée de votre serveur, vous aurez bientôt le privilège de leur envoyer un chèque dès l'achat de votre prochain PC, sans même l'avoir sorti du carton. Aucun prix n'a encore été fixé pour cette nouvelle taxe, inspirée sans doute de l'actualité allemande. Depuis le premier janvier, en effet, une taxe d'environ 30 euros y est de rigueur. Rappelons que la motivation première de cette mesure est d'obtenir une compensation pour les artistiques lésés de leurs droits par la diffusion de leurs œuvres. Cela concerte en premier

chef l'industrie musicale, qui comme chacun le sait, subit les sept plaies d'Égypte que sont les Napster, Freenet ou autre Gnutella... Internet génère de nouveaux comportements, parfois peu compatibles avec les habitudes précédentes. Voilà un beau procès d'intention fait aux utilisateurs de micro-informatique, puisque le simple achat de matériel fait de vous un pirate en puissance ! À moins qu'il ne s'agisse ni plus ni moins d'une légalisation du piratage ?

Ce facétieux XXI^e siècle nous aura fait passer de l'ère de la vignette auto à celui de la taxe réseau, tout un symbole finalement. Chapeau l'artiste !

Dernière minute : nous venons d'apprendre que le projet vient d'être abandonné en rase campagne, suite à l'intervention de Laurent Fabius, ministre des Finances, qui voyait d'un très mauvais œil ce projet de taxe en pleine période de réduction d'impôts. ●

<http://www.april.org/articles/communiques/pr-tax.html>

<http://www.vachealait.com>

Profits Warning

Résultats décevants pour les fabricants d'ordinateurs

Les uns après les autres, tous les fabricants d'ordinateurs personnels font leurs « *profits warning* ». Les résultats du quatrième trimestre 2000 sont en-deçà des prévisions. Ainsi, Compaq, Gateway, HP ou Dell ont déjà prévenu leurs actionnaires que les prévisions ne seraient pas tenues. Apple, pour sa part, connaît sa première perte en trois ans, inférieure tout de même à ce que la rumeur prévoyait. Il faut préciser que le taux d'équipement des ménages américains en matière de PC a dépassé les 50 % cette année. De fait, l'industrie qui jusque-là s'était habituée à une croissance annuelle à deux chiffres se retrouve désormais

confrontée à un marché de renouvellement, d'autant que l'Europe ne devrait plus tarder à rejoindre les mêmes taux. Toutefois, certains acteurs majeurs du marché informatique connaissent de grands succès cette année, IBM tout d'abord, qui fait état d'une hausse de 28 % de son résultat net et d'un bénéfice net de 2,7 milliards de dollars. Pour SUN, c'est également le plein soleil, puisque la société annonce une hausse de 40 % de son chiffre d'affaire, même si les analystes doutent qu'une telle croissance puisse être maintenue très longtemps. L'année 2000 aura surtout été profitable du côté des serveurs. ●

■ AMD prépare l'arrivée de son 64 bits

En collaboration avec la société Virtutech, AMD vient de sortir une suite logicielle dont la fonction est d'émuler la future architecture 64 bits du fondeur. Ce futur processeur, pour l'instant connu sous le nom de code de « *Hammer* » devrait voir le jour vers le milieu de l'année 2002. Grâce au « *virtuHammer* », les développeurs pourront d'ores et déjà préparer leur passage vers cette plate-forme en utilisant cet émulateur sur un Athlon 32 bits classique.

■ DivX ouvre ses sources

Comme promis, l'équipe de Project Mayo vient d'ouvrir les sources du fameux codec vidéo DivX (attention toutefois, les restrictions de la licence font qu'il ne s'agit pas d'un logiciel libre). Le code est disponible sur le site web, sous le nom d'OpenDivX. Ce dernier est basé sur les standards MPEG-4. Ceci devrait amenuiser considérablement l'intérêt porté aux codecs concurrents, d'autant que DivX est déjà très répandu (notamment auprès des pirates de DVD vidéo). Seule faiblesse de DivX, il ne permet pas de gérer une diffusion en ligne, ce qui l'empêchera sans doute de devenir un standard de facto. Par contre, le nouveau projet de Project Mayo, DivX Deux, permettra le « *streaming* » et améliorera les performances générales du codec à partir d'OpenDivX. Il n'est pas sûr, par contre, que les sources soient alors disponibles... Alors ? DivX deviendra-t-il le MP3 de la vidéo ? Le marché paraît encore trop jeune pour répondre avec certitude. <http://www.projectmayo.com/>

Systemes embarqués : optimisez !


 Sur votre CD-Rom

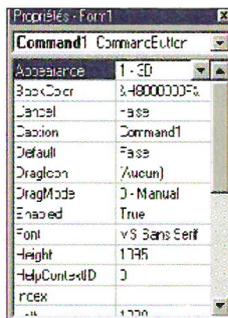

Le constructeur d'assistants personnels PALM (en association avec l'éditeur AppForge) a décidé de s'attirer les bonnes grâces des millions de développeurs Visual Basic en proposant un Kit de développement VB pour PalmOS.

Xavier Leclercq
Xavier.Leclercq@programmez.com

Nous connaissons aujourd'hui avec les systèmes embarqués, le même phénomène que l'on a pu observer hier, en passant d'un environnement tel que DOS à Windows 3.x ou encore de win3.1 à Windows 95 puis à NT4 et W2K. Les contraintes de mémoire et le mode graphique ont pour conséquence d'affubler Visual Basic « d'une certaine lenteur ».

Par exemple, sous Windows NT 4, les composantes GDI (*Graphics Device Interface*) et USER (gérant l'affichage des fenêtres) ont été réécrits en mode privilégié (ce qu'on nomme parfois le mode « noyau » ou encore le mode « executive »). Le gain résultant pour des applications telles que VB, qui peut employer de manière intensive des graphismes, est de l'ordre de 10%. Mais en moyenne, on a pu observer que le gain de performances compensait à peine le passage de l'interface Windows NT 3.x au système G.U.I. de NT4.

Ces réflexions suscitent des questions. Quelle influence peut avoir le sous-système graphique de mon système d'exploitation sur mon application VB? Comment améliorer discrètement mais sûrement les performances ?



> La propriété Appearance en Visual Basic 6 permet en théorie (et non en pratique) de visualiser un contrôle sous la forme 3D ou 2D (plate).

Selon votre système d'exploitation

Sous VB 6, il n'est pas possible de passer d'un bouton 3D à un bouton 2D plat. À première vue, le rapport qui résulte de cette constatation et celui des performances de votre application n'est pas évident à établir. Essayez d'abord la manipulation. Même en modifiant la propriété Appearance du bouton n'y change rien.

Est-ce un bug de VB 6? Non. Sous Windows 3.x une DLL du nom de CTL3D permettait au système d'exploitation de simuler l'effet visuel de l'aspect 3D. Le sous-système graphique quelle que soit la plate-forme teste-

ra d'abord au moment de l'exécution la version du système d'exploitation (`GetVersion()` et `GetVersionEx()`). Puis si le numéro majeur est inférieur à 4 (donc à Windows 95) la DLL CTL3D pourra être sollicitée. Sinon le sous-système graphique emploiera la fonction `GDI DrawEdge()`, et ce, quel que soit l'état du style réclamé par un contrôle, 3D ou NON 3D...

Première réponse : selon le système d'exploitation l'effet 3D n'est pas géré de la même manière. Et exiger de la 3D sollicite pas mal de temps CPU!

Du 3D en Basic

Ce sujet du «3D» est aussi discuté sur les forums de discussion VB, et parfois, des idées germent de l'esprit de certains développeurs. Par exemple, la courte routine suivante proposée par M. Nastran, transforme un contrôle 2D en 3D à condition que la propriété **Tag** de ce contrôle soit fixée à «3D» (vous devez placer la routine dans Form Paint).

```
Sub form3d (formname As form)
  Dim drkgray As Long, fullwhite As Long
  Dim i As Integer
  Dim ctop As Integer, clef As Integer, cright As Integer, cbottom As Integer

  Dim cname As Control

  drkgray = RGB(128, 128, 128)
  fullwhite = RGB(255, 255, 255)

  dw = formname.DrawWidth
  formname.DrawWidth = 1
  For i = 0 To (formname.Controls.Count - 1)
    Set cname = formname.Controls(i)
```

```
If TypeOf cname Is Menu Then
  'Debug.Print "menu item"
Elseif InStr(UCCase(cname.Tag), "3D") <> 0 Then
  ctop = cname.Top - Screen.TwipsPerPixelY
  clef = cname.Left - Screen.TwipsPerPixelX
  cright = cname.Left + cname.Width
  cbottom = cname.Top + cname.Height
  formname.Line (clef, ctop)-(cright, ctop), drkgray
  formname.Line (clef, ctop)-(clef, cbottom), drkgray
  formname.Line (clef, cbottom)-(cright, cbottom), fullwhite
  formname.Line (cright, ctop)-(cright, cbottom), fullwhite
End If
Next i
formname.DrawWidth = dw
End Sub
```

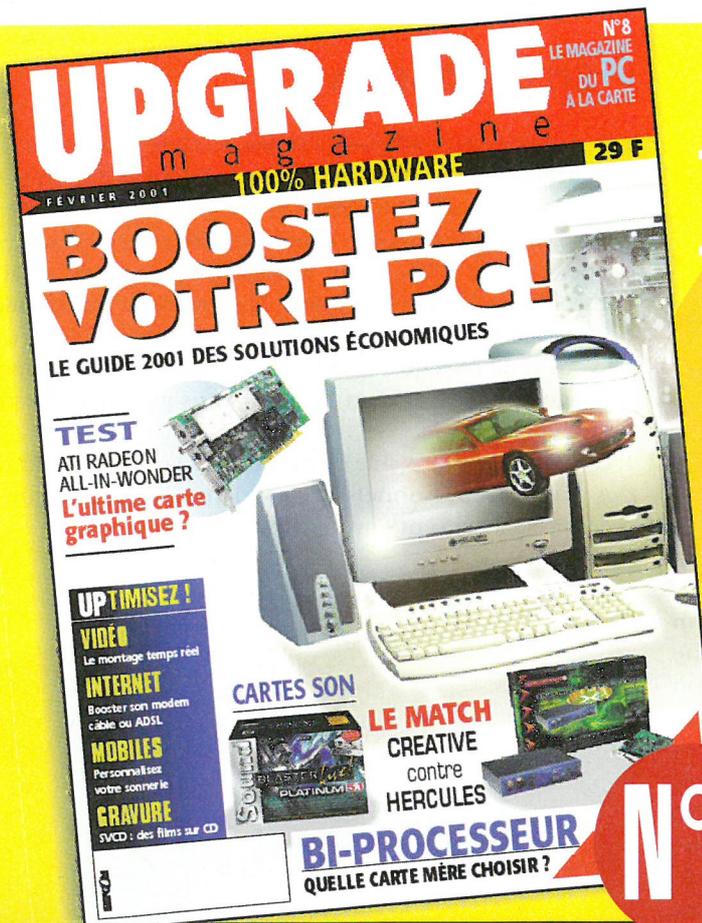
Ou encore, voici le code proposé par M. Dobzinski qui permet de passer d'un «look 3D couleur» à un «look monochrome» :

La partie des déclarations : (voir listing 1)

Le code proprement dit : (voir listing 2)

Belles performances mais que de temps CPU consacré à cette manœuvre!

```
Declare Function GetPixel Lib "GDI" (ByVal hdc As Integer, ByVal X As Integer, ByVal Y As Integer) As Long
Declare Function SetPixel Lib "GDI" (ByVal hdc As Integer, ByVal X As Integer, ByVal Y As Integer, ByVal crColor As Long) As Long
```



Nouveau !

Un rendez-vous
à ne pas
manquer

N°8

En vente actuellement chez votre marchand de journaux

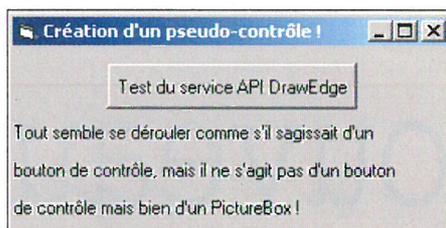
Une première réflexion s'impose : lorsque vous avez de nombreux objets à dessiner sur une même feuille, placez-les dans un objet «container» que vous rendrez d'abord invisible. Puis basculez à **TRUE** la propriété visible de ce «container» afin que l'ensemble ne soit dessiné qu'une seule et unique fois !

Une API très spéciale

Sous Windows 2000/NT4 ou 9x, les choses se compliquent. Comment obtenir la parfaite maîtrise de l'aspect 3D ? Comme nous l'avons vu, il n'est pas possible de basculer la propriété **Appearance** et d'en constater l'effet. Par exemple, sous Windows 9x, c'est le service **API DrawEdge** qui se charge de la besogne ignorant le drapeau 3D ou non 3D de vos contrôles.

Consultez, sur le **listing 3** dans votre CD-Rom d'accompagnement, les explications fournies par Microsoft au sujet de cette fonction **DrawEdge**.

Le code en VB 6, que je vous propose au **listing 4** intitulé «**Module DrawEdge.BAS**» est passionnant. Il gère lui-même la fonction **DrawEdge** en créant de toute pièce un contrôle à partir d'un «Picture Box». Examinez d'abord le code.



> Le bouton «Test du service API DrawEdge» est en réalité un contrôle «PictureBox» simulant parfaitement l'apparence d'un bouton de commande grâce au service **API DrawEdge** (API qui est employé par Windows 9x pour dessiner des contrôles en 3D).

Cette approche par attaque directe de la fonction **DrawEdge** n'est pas sans avantages :

- Vous pouvez basculer d'un aspect 3D à 2D très simplement sans rien reprogrammer.

C'est pratique pour un utilisateur qui est encore doté d'un écran monochrome (espèce en voie de disparition, mais que l'on rencontre encore dans l'industrie, faute de budget), ou encore pour imprimer un formulaire sur une imprimante noir et blanc.

- C'est aussi une manière pour gagner du temps CPU, car un contrôle classique ou *via* une programmation directe de **DrawEdge** accapare un temps de traitement non négligeable.
- Ce procédé intéresse également le monde de «l'embarqué» où la moindre seconde, le moindre octet gagné se révèle important.

```
Sub Enable3DButton (Cmd3D As Control, SrcPicture As Control, Enable As Integer)
'Cmd3D : un contrôle 3D
'SrcPicture : un PictureBox employé comme tampon intermédiaire pour le traitement.
'Enable : comme la propriété Enabled.
```

```
Dim Color As Long, RetVal As Long, X As Integer, Y As Integer
SrcPicture.AutoSize = True
SrcPicture.ScaleMode = 3
SrcPicture.AutoRedraw = True
```

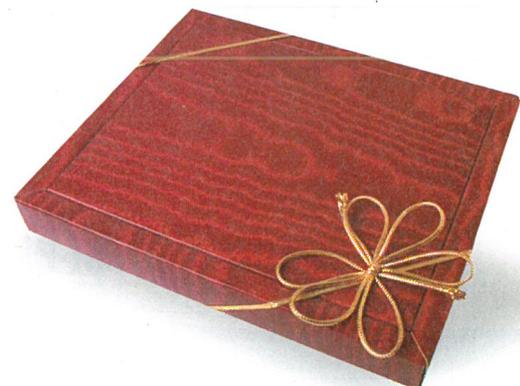
```
Cmd3D.Enabled = Enable
If Not Enable Then
SearchColor = 0
ReplaceColor = 8
Else
SearchColor = 8
ReplaceColor = 0
End If
SrcPicture.Picture = Cmd3D.Picture
For Y = 0 To SrcPicture.ScaleHeight - 1
For X = 0 To SrcPicture.ScaleWidth - 1
Color = GetPixel(SrcPicture.hDC, X, Y)
If Color = QBColor(SearchColor) Then
RetVal = SetPixel(SrcPicture.hDC, X, Y, QBColor(ReplaceColor))
End If
Next
Next
Cmd3D.Picture = SrcPicture.Image
End Sub
```

2

En conclusion

Un contrôle «3D» affiché à l'écran accapare un temps CPU non négligeable (à l'initialisation et à chaque **REFRESH** si **AutoRedraw()** est à **TRUE**). Pour éviter cela, il est possible de réaliser un «**print screen**» d'un contrôle puis de récupérer l'aspect visuel «3D» sous **PaintBrush** et sauver ainsi le contrôle sous format **BMP**. Ainsi, le contrôle se transformera en simple «**Picture Box**» qui n'emploie pas **DrawEdge()**. Il ne vous reste plus qu'à construire une sorte de bibliothèque d'images de contrôles en 3D pour vous y plonger lorsque cela sera nécessaire.

Il est maintenant possible de développer en Visual Basic avec des assistants personnels et c'est dans ce contexte que de telles ficelles peuvent s'avérer très utiles. ■



Faites vos Applets sur internet



Donner de la vie à des sites web n'est pas si difficile qu'il y paraît. Un rien d'astuce, un peu de Java, et voila des pages qui s'éveillent.

Par Laurent FOYNARD et Christophe BABAYOU
contact@uplog.com

Le Java, à cause de la sur-couche imposée par la machine virtuelle nécessaire à son bon fonctionnement, est un langage bien plus lent que le C++ (d'environ un facteur quatre). De ce fait, il est bien moins adapté que ce dernier à la création de jeux vidéo et d'animations. Il y a cependant un environnement où le Java sait montrer toute son envergure, je veux bien entendu parler d'Internet. Nous allons vous exposer toutes les bases indispensables à la réalisation d'animations et de jeux vidéo à l'intérieur d'applettes, à partir du JDK 1.2.

Mais pourquoi une applet, et pourquoi le JDK 1.2 ?

La réponse est des plus évidentes. Il n'y a rien de plus triste qu'un site web plat et banal. Les *applets* sont un formidable moyen d'apporter de la vie sur vos pages html. Grâce à elles et à cet article, vous pourrez proposer à vos visiteurs des animations graphiques et vidéo, ainsi que des jeux simples, mais amusants. Si avec tout cela, la fréquentation de votre site web n'augmente pas, vous devrez vous poser de très grosses questions...

Quant au choix de la JDK1.2, la raison en est que c'est la ver-

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class affiche_image extends Applet {
    Image img;

    public void paint(Graphics g)
    {
        g.drawImage(img,0,0,this);
    }

    public void init() {
        img = getImage(getCodeBase(),»img.jpg»);
    }
}
```



sion la plus employée par les navigateurs web actuels, et qu'ainsi, vous garanzissez une compatibilité maximale avec les machines de vos visiteurs.

Je tiens cependant à vous préciser une chose avant d'entrer dans le vif du sujet, ne vous attendez pas à des résultats spectaculaires. Le Java est déjà plutôt lent, et les *applets* le sont encore plus. Elles n'en demeurent pas moins capables, sans aucun problème, de gérer les déplacements et les actions de plusieurs *sprites*, ainsi que de l'inévitable décor, dans un environnement 2D.

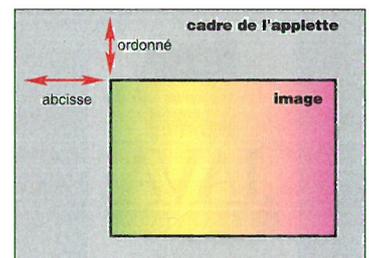
Gestion graphique

Les bases

Dans les *applets* la gestion graphique est assurée par deux fonctions nommées `getImage` et `drawImage`. Comme vous vous en doutez certainement, la première a pour fonction de charger l'image dans une variable, et la seconde, celle de l'afficher. Avant d'aller plus loin, il convient d'étudier plus en détail ces deux fonctions. Pour ce faire, décortiquons le code d'une *applet* affichant simplement une image (voir le [listing 1](#)).

La fonction `Paint` est automatiquement appelée par l'*applet*, lors de son affichage, et lors de toute modification liée à la taille de la fenêtre ou, à sa position.

L'objet de type «*Graphics*» qu'elle propose est un lien direct vers la sortie écran. Tout ce qui est écrit sur cet objet l'est automatiquement à l'écran. C'est ce qu'on fait avec la fonction `g.drawImage(img,0,0,this`. Les paramètres deux et trois de cette fonction (tous les deux égaux à zéro) correspondent respectivement à l'abscisse et à l'ordonnée du point supérieur gauche de cette image. Si on change leurs valeurs, on change l'emplacement de l'image à l'écran (vous voyez l'intérêt de cette fonction...).



La fonction permettant de charger l'image dans l'appliquette se trouve dans la fonction `init` (fonction qui, je le rappelle, est automatiquement appelée au démarrage de l'*applet*). Il s'agit de la fonction suivante : `img = getImage(getCodeBase(),«img.jpg»);`.

`getCodeBase` est une fonction qui retourne l'URL où se trouvent les classes de l'*applet*. Nous aurions pu également utiliser la fonction `getDocumentBase` afin de récupérer l'URL de la page web qui a chargé l'*applet*. «img.jpg» est, bien entendu, le nom de l'image à charger. Cette dernière peut être soit au format jpg, soit au format gif. Selon que l'on utilise `getCodeBase` ou `getDocumentBase`, il faut veiller à ce que l'image soit dans le même répertoire que la page web, ou que les classes de l'*applet*. Bien souvent, tout se trouve au même endroit et l'on n'a pas de question à se poser.

Un peu plus loin...

Maintenant que vous savez afficher une image à l'écran, et avant de faire notre première animation, il convient de connaître la manière de contrôler le rafraîchissement de l'affichage.

Bien que la fonction `paint()` soit automatiquement appelée par l'*applet*, il existe cependant une méthode pour provoquer son instanciation n'importe quand.

Voici le code permettant d'effectuer cela :

```
repaint();

public void update(Graphics g)
{
    paint(g);
}
```

La fonction `repaint()` peut être utilisée n'importe où dans votre code, elle appelle la méthode `update`, qui elle, peut instancier la méthode `paint`. Comme dans cette dernière, l'objet «Graphics» de la fonction `update` pointe sur l'écran. Nous avons maintenant tous les outils en main pour réaliser notre première animation.

Première animation

Nous allons créer cette animation, en n'utilisant qu'une seule image (ce n'est donc pas une véritable animation), que l'on déplacera dans l'espace. Pour ce faire, nous sommes partis du code précédemment fourni et y avons ajouté quelques éléments. (voir [listing 2](#))

Comme vous pouvez le constater, les changements ne sont pas légion. Nous avons instancié un *thread*, afin de rendre le déplacement de l'image «cycle» et nous avons créé une variable «abs», qui permet de faire varier l'abscisse de l'image. Le *thread* incrémente l'abscisse de l'image, puis appelle la fonction d'affichage. Cette dernière utilise la méthode `clearRect` pour effacer l'image précédemment inscrite à l'écran (vous pourrez constater de son intérêt en lançant l'*applet* sans cette ligne).

2

```
public class affiche_image extends Applet implements Runnable{
    Image img;
    int abs;
    Thread T;

    public void paint(Graphics g) {
        g.drawImage(img,abs,0,this);
    }

    public void init() {
        img = getImage(getCodeBase(),«img.jpg»);
        T=new Thread(this);
    }
    T.start();
}

public void run() {
    while(true)
    {
        for(abs=0;abs<1000;abs++)
        {
            try{T.sleep(100);}catch(Exception e){}
            repaint();
        }
    }
}

public void update(Graphics g)
{
    g.clearRect(abs-1,0,img.getHeight(this),img.getWidth(this));
    paint(g);
}
```

Faites varier la valeur du `sleep` se trouvant dans le *thread* pour modifier le temps de présence à l'écran de chaque image. Comme vous pourrez le constater, plus ce temps est faible, plus les images clignotent à l'écran. Cela est dû à leur temps d'affichage. Rassurez-vous, il existe une méthode aussi simple qu'efficace pour se débarrasser de ce problème, le *double buffering*, qui sera présenté un peu plus loin dans cet article.

Maintenant que nous savons déplacer une image, nous allons effectuer notre première véritable animation.

Une véritable animation

La première chose à faire est de créer une suite d'images sensées décomposer le déplacement d'un personnage. Nous allons les nommer de «1.jpg» à «5.jpg». Le code ne va souffrir que de peu de changements, comme vous pouvez le constater dans le [listing 3](#).

Afin de stocker leurs références, nous avons fait appel à un tableau d'images, que nous remplissons dans la fonction `init()`. Une fois cela fait, il suffit d'afficher les images les une après les autres. Nous avons pour cela utilisé une variable nommée «nbr_img», qui contient le numéro de l'image à afficher. Cette variable est incrémentée après chaque affichage, et régulée par un modulo 5, afin de s'assurer que sa valeur sera toujours entre 0 et 4 (à savoir les numéros des cases du tableau contenant les références des images à afficher). C'est tout.

3

```

public class affiche_image extends Applet implements Runnable{
    Image img[]=new Image[5];
    int nbr_img=0;
    int abs;
    Thread T;

    public void paint(Graphics g)
    {
        g.drawImage(img[nbr_img],abs,0,this);
    }

    public void init() {
        for(int i=1;i<6;i++)
            img[i-1] = getImage(getCodeBase(),i+«.jpg»);
        T=new Thread(this);
        T.start();
    }

    public void run()
    {
        for(abs=0;abs<1000;abs++)
        {
            try{
                T.sleep(100);
            }catch(Exception e){}
            repaint();
            nbr_img=(nbr_img+1)%5;
        }
    }

    public void update(Graphics g)
    {
        g.clearRect(abs-1,0,img[nbr_img].getHeight(this),img[nbr_img].getWidth(this));
        paint(g);
    }
}

```

Double buffering

Comme vous vous en êtes très certainement aperçu dans les exemples précédents, les animations provoquent des clignotements (nommé *clipping* en anglais) très désagréables, qui ont la fâcheuse tendance à devenir de plus en plus présents, jusqu'à rendre les images parfaitement invisibles à l'œil humain. Cela est bien entendu un énorme handicap, mais peut être heureusement fort aisément contourné en utilisant une simple méthode de *double buffering* (tampon double, en français).

Reportez-vous au **listing 4**

pour y voir la méthode appliquée à notre exemple précédent. Pour réaliser le double tampon, il suffit d'ajouter deux nouveaux objets. L'un de type «Image», et l'autre de type «Graphics».

Dans la fonction `init()` nous utilisons la méthode `createImage` pour instancier notre objet «Image», nommé «tampon_img». La taille de cette image doit correspondre à la taille de la partie graphique de l'*applet*. Dans la ligne qui suit, nous lions notre image à notre objet «Graphics» nouvellement créé *via* la méthode `getGraphics()`.

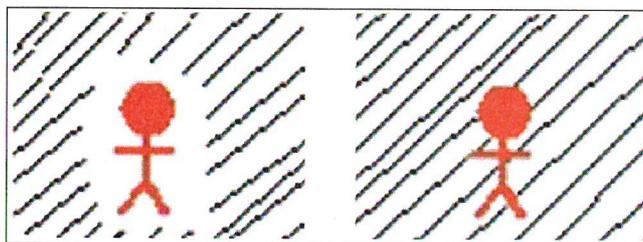


Image dont le fond n'est pas transparent.

Image .gif dont le fond a été rendu transparent

Le lien ainsi créé est une sorte d'écran virtuel. Tout ce que nous écrivons par la suite dans l'objet «Graphics», nommé tampon, ne sera stocké qu'en mémoire. Mais, grâce à l'objet «Image» qui lui est associé, nous pourrons afficher à l'écran tout ce que contient cet écran virtuel.

Tout le travail du tampon se passe dans la fonction `update`. On commence par effacer l'intégralité de l'écran virtuel, en utilisant la méthode `clearRect` utilisée précédemment, remarquez que cette fonction n'est pas utilisée sur l'objet «g», mais bien sur l'objet «tampon», correspondant à notre écran virtuel. Nous appelons ensuite la méthode `paint()`, mais en passant en paramètre «tampon», et non «g». Comme cela, tout ce qui sera inscrit par cette méthode `paint()` le sera en mémoire et non à l'écran. Une fois que tout cela est fait, et que l'écran virtuel contient tout ce qui doit figurer à l'écran, on peut provoquer son affichage, en utilisant la fonction `drawImage` sur l'objet «g», en passant l'objet «tampon_img» en paramètre (n'oubliez pas que cet objet est lié à l'écran virtuel).

En comparant les animations résultantes de ce code et du précédent, vous réaliserez combien l'apport du *double buffering* est indispensable. D'autant que, sans lui, plus il y a d'images à l'écran, plus le phénomène de clignotement est présent.

Fond d'écran

L'habillage de votre jeu est un élément essentiel si vous tenez à faire quelque chose d'esthétique. Un fond d'écran est un habillage indispensable pour tout jeu digne de ce nom. Son affichage et sa gestion ne diffèrent pas de ce que nous avons vu précédemment.

On peut afficher plusieurs images à l'écran (et aussi dans l'écran virtuel), en appelant plusieurs fois d'affilée la fonction `drawImage` sur le même objet graphique. Cela permet de dessiner le fond d'écran et plusieurs personnages. Il est à noter que les dessins se posent les uns au-dessus des autres,

au fur et à mesure de leur affichage, de la même manière que si l'on empilait des feuilles de papier.

De ce fait, si vous modifiez le code précédemment dévoilé, afin d'insérer un fond d'écran, vous constaterez que le personnage est entouré d'un cadre qui se superpose au fond

d'écran. Cela est dû au fond de l'image du personnage, qui écrase ce qui se trouve derrière. Pour corriger ce problème, il suffit de transformer les images du personnage au format gif, et de préciser que la couleur de fond est transparente. Vous pouvez réaliser cette opération avec n'importe quel bon logiciel de dessin du type Paint Shop Pro ou Photoshop. Vous constaterez alors que le cadre autour du personnage disparaît de l'écran. Cela fonctionne exactement de la même manière que les fonds bleu ou vert au cinéma et à la télévision. D'ailleurs, tout comme eux, il ne faut pas que le per-

sonnage « porte sur lui » la même couleur que le fond de l'image, car cette partie de sa personne disparaîtra pour faire place au fond d'écran.

code permettant d'afficher un fond d'écran, et une autre image

```
public void paint(Graphics g)
{
    g.drawImage(image_de_fond,0,0,this);
    g.drawImage(imge_en_premier_plan,10,10,this);
}
```

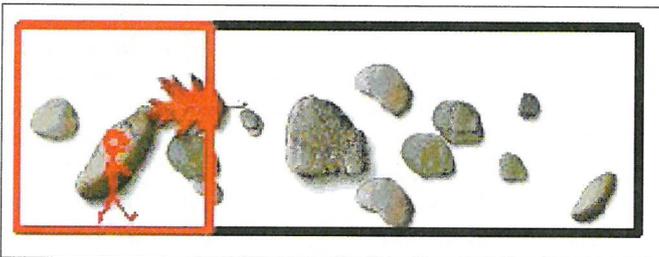
Scrolling

Maintenant que nous savons superposer plusieurs images (donc un personnage sur un décor), nous allons nous attaquer à un sujet encore plus intéressant : le *scrolling*.

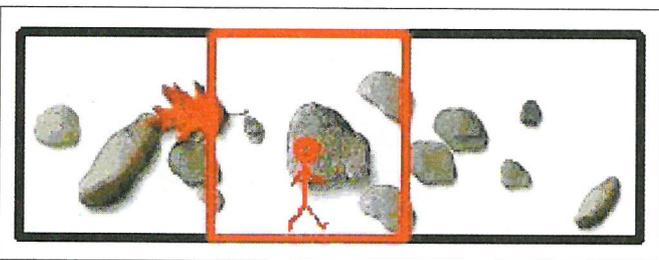
Qu'il soit horizontal ou vertical (voire les deux, le *scrolling* obéit à la même loi, ce n'est pas le personnage qui se déplace, mais tout ce qui l'entoure.

Les différences entre les animations précédemment réalisées et un *scrolling* viennent juste du fait qu'il y a deux images présentes à l'écran, un fond et un personnage ; ce dernier reste fixe à un endroit de l'écran, et c'est le fond qui se déplace.

Bien entendu, le décor se déplace dans le sens opposé de celui du personnage. Si le personnage va à droite, le fond d'écran doit se décaler vers la gauche.



Abscisse de l'image dans la fonction drawImage vaut zéro.



Abscisse de l'image dans la fonction drawImage est négatif.

Pour réaliser un bon *scrolling*, il faut une image de décor bien plus grande que l'écran de l'*applet*. L'utilisateur ne verra que la partie apparente de l'image, mais la mémoire nécessaire à l'ensemble de l'image sera tout de même utilisée (attention donc aux ressources). Il ne reste plus qu'à faire varier l'abscisse du décor dans le cas d'un *scrolling* horizontal, et l'ordonnée dans le cas d'un vertical (voire les deux).

Il est important de noter que la fonction `drawImage`, les paramètres abscisse et ordonnée peuvent être négatifs. Si

```
public class affiche_image extends Applet implements Runnable{
    Image img[]=new Image[5];
    int nbr_img=0;
    Graphics tampon;
    Image tampon_img;
    int abs;
    Thread T;
    public void paint(Graphics g)
    {
        g.drawImage(img[nbr_img],abs,0,this);
    }
}
```

```
public void init() {
    for(int i=1;i<6;i++)
        img[i-1] = getImage(getCodeBase(),i+».jpg»);
    tampon_img= createImage (400,400);
    tampon= tampon_img.getGraphics();
    T=new Thread(this);
    T.start();
}
public void run()
{
    for(abs=0;abs<1000;abs++)
    {
        try{
            T.sleep(100);
        }catch(Exception e){}
        repaint();
        nbr_img=(nbr_img+1)%5;
    }
}
public void update(Graphics g)
{
    tampon.clearRect(0,0,400,400);
    paint(tampon);
    g.drawImage(tampon_img,0,0,this);
}
}
```

4

l'abscisse et l'ordonnée ont pour valeur zéro, le coin supérieur gauche de l'image sera pile au début de l'écran. Si l'abscisse devient négative, le début de l'image se décalera vers la gauche (ce qui provoquera donc le *scrolling*), et disparaîtra de l'écran tandis qu'une partie encore invisible fera son apparition.

Conclusion

Vous avez maintenant tous les outils de base nécessaires à la réalisation d'animations. Bien entendu, il existe de nombreuses fonctions que nous n'avons pas abordées (comme certaines permettant de découper des parties d'images), ce qui pourra faire l'objet d'autres articles. ■

Auteurs :

Christophe Babayou et Laurent Foynard
Ingénieurs chez UPLOG
contact@uplog.com

La sécurité Java 2

Une première approche



L'internet permet une multiplication des moyens de diffusion des applications et donc des risques d'accès indésiré à vos données.

Nicolas Dasriaux

Le courrier électronique, le Web, le FTP sont autant de moyens de distribuer des composants exécutables, qu'il s'agisse d'*applets*, d'ActiveX ou d'applications. Chacun de ces composants exécutables est susceptible d'agir de manière hostile, une fois installé sur la machine hôte.

Le système d'exploitation offre, la plupart du temps, le moyen de réduire l'accès aux fichiers ou aux ressources de la machine. Cependant le composant exécutable a, le plus souvent, les mêmes droits que l'utilisateur qui l'exécute. Ceci permet donc au composant d'accéder à tous les fichiers de l'utilisateur et d'en faire ce qu'il en veut : effacement, modification, envoi vers une autre machine, etc.

On le voit, les systèmes d'exploitation ne permettent pas un niveau de sécurité suffisant.

Idéalement, il devrait être possible de surveiller l'exécution d'un composant exécutable afin de l'empêcher d'agir de manière hostile. Cet idéal existe quand le composant est exécuté par une machine virtuelle Java 2. Dans ce cas, il est possible de définir finement les permissions accordées au composant, par exemple : les fichiers qu'il a le droit de lire ou de modifier, la possibilité d'accéder au presse-papier, la possibilité d'accéder à l'imprimante.

L'auteur :

Nicolas Dasriaux est consultant chez Neoxia, un cabinet de conseil en architectures distribuées.



Installation des exemples

Pour installer les exemple de cet article, recopiez le dossier `security` du CD-ROM à la racine de votre disque dur C:. Veillez à ce que le répertoire bin du JDK (au minimum 1.2.2, de préférence 1.3) soit dans le `PATH`, ceci vous permettra d'accéder à `java.exe`, `appletviewer.exe` et `policytool.exe`.

Tous les exemples seront lancés à l'aide de la ligne de commande.

Exemple d'une applet

Prenons le cas d'une *applet* simple, que nous allons décrire. Elle permet de saisir le chemin d'un dossier et un mot. En appuyant sur un bouton, on crée, dans le dossier saisi, le fichier `Word.txt` qui contient le mot saisi.

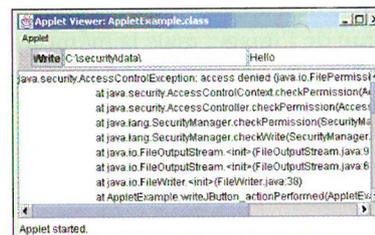
Tous les fichiers nécessaires au bon déroulement de l'exemple se trouvent dans le répertoire `C:\security\applet`:

- `AppletExample.java` contient le code source de l'*applet* (voir le [listing 1](#)),
- `AppletExample.html` est la page web qui accueille l'*applet* (voir le [listing 2](#)).

Pour exécuter l'*applet*, lancer la fenêtre de ligne de commande («fenêtre DOS»), se déplacer dans le dossier `C:\security\applet`, et lancer `AppletExample1.bat` (voir le [listing 3](#)).

Listing 3 : AppletExample1.bat

```
appletviewer file:///C:/security/applet/AppletExample.html
```



> Figure 1 : exécution de `AppletExample1.bat`.

Appuyer sur le bouton *Write*, une exception est alors immédiatement levée et apparaît dans la zone texte (voir la figure ci-contre).

Listing 1 : AppletExample.java

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;

public class AppletExample extends JApplet {
    JToolBar jToolBar = new JToolBar();
    JButton writeJButton = new JButton();
    JTextField folderJTextField = new JTextField();
    JTextField wordJTextField = new JTextField();
    JScrollPane logScrollPane = new JScrollPane();
    JTextArea logJTextArea = new JTextArea();

    public AppletExample() {
    }

    public void init() {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            jToolBar.init();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jToolBarInit() throws Exception {
        this.setSize(new Dimension(600, 300));
        writeJButton.setText("Write");

        writeJButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                writeJButton_actionPerformed(e);
            }
        });

        folderJTextField.setText("C:\\security\\data\\");
        wordJTextField.setText("Hello");
        this.getContentPane().add(jToolBar, BorderLayout.NORTH);
        jToolBar.add(writeJButton, null);
        jToolBar.add(folderJTextField, null);
        jToolBar.add(wordJTextField, null);
        this.getContentPane().add(logScrollPane, BorderLayout.CENTER);
        logScrollPane.getViewPort().add(logJTextArea, null);
    }

    void writeJButton_actionPerformed(ActionEvent e) {
        try {
            logJTextArea.setText("");

            Writer w = new FileWriter(folderJTextField.getText() + "Word.txt");
            w.write(wordJTextField.getText());
            w.close();

            logJTextArea.append(
                "Word '" + wordJTextField.getText() +
                "' was successfully written in file '" +
                folderJTextField.getText() + "Word.txt'."
            );
        }
        catch (Exception ex) {
            Writer w = new StringWriter();
            PrintWriter pw = new PrintWriter(w);
            ex.printStackTrace(pw);
            logJTextArea.append(w.toString());
            pw.close();
        }
    }
}

```

Listing 2 : AppletExample.html

```

<html>
<head>
  <meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
  <title> Applet Example </title>
</head>
<body>
  <h1> Applet Example </h1>
  <applet
    codebase = "file:///C:/security/applet/"
    code = "AppletExample.class"
    name = "TestApplet"
    width = "600"
    height = "300"
    hspace = "0"
    vspace = "0"
    align = "middle">
  </applet>
</body>
</html>

```

```

java.security.AccessControlException: access denied
(java.io.FilePermission C:\security\data\Word.txt write)

```

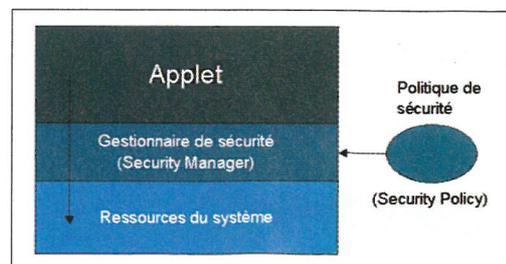
Cette exception indique que l'*applet* n'a pas la permission d'écrire le fichier `Word.txt` dans le dossier `C:\security\data\`. L'exception est précisément levée au moment de l'appel du constructeur dans la ligne de code suivante :

```
Writer w = new FileWriter(folderJTextField.getText() + "Word.txt");
```

Gestionnaire de sécurité

En fait, toute *applet* est exécutée sous le contrôle de ce que l'on appelle un gestionnaire de sécurité ou *Security Manager*. Le gestionnaire de sécurité s'interpose entre l'*applet* et les ressources du système. Il autorise ou n'autorise pas l'accès aux ressources du système en fonction de ce que l'on appelle une politique de sécurité ou *Security Policy* (voir la [figure 2](#)).

Une politique de sécurité définit les permissions accordées au code (On entend ici par code, l'ensemble des `.class` de l'*applet*).



> Figure 2 : gestionnaire de sécurité et politique de sécurité.

Quand une *applet* tente d'accéder à une ressource du système, le gestionnaire de sécurité vérifie que le code a la permission d'accéder à la ressource en consultant la politique de sécurité. Si la permission est accordée, tout se passe nor-

malement, dans le cas contraire, une exception `AccessControlException` est levée. C'est ce qui se passe dans notre exemple.

Politique de sécurité

Une politique de sécurité se compose d'un ensemble de fichiers `.policy` au format texte. Chacun d'entre eux contient un ensemble de permissions qui spécifient ce qu'un code a le droit de faire en fonction de sa provenance (URL du code, signataire du code). Lorsqu'aucune politique de sécurité n'est spécifiée, le gestionnaire de sécurité utilise la politique de sécurité par défaut, plus connue sous le nom de bac à sable ou *Sandbox*. Le bac à sable est une politique de sécurité très restrictive qui garantit que l'*applet* ne pourra pas avoir de comportement hostile. Entre autre, le bac à sable empêche l'accès à toutes les ressources de la machine locale. En revanche, le bac à sable autorise l'*applet* à accéder à la machine depuis laquelle elle a été téléchargée.

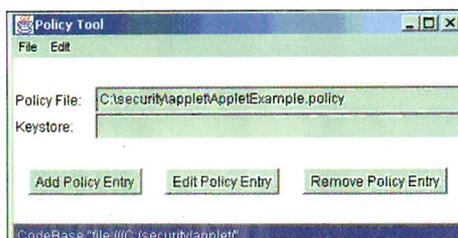
Dans le cas de notre *applet*, c'est bien le bac à sable qui est la politique de sécurité active. De manière générale, la politique de sécurité active est déduite de trois fichiers principaux : `java.security`, `java.policy`, `java.policy` :

- le fichier `java.security` situé dans le dossier `lib\security\` du JDK. Il énumère les fichiers `.policy` pris en compte pour constituer la politique de sécurité active. S'il n'a pas été modifié, il fait référence à `java.policy` et `java.policy`.
- le fichier `java.policy` situé également dans le dossier `lib\security\` du JDK. Pour simplifier, il participe à la description des permissions du bac à sable.
- `java.policy` s'il est situé dans le dossier de l'utilisateur en cours est facultatif. En revanche, s'il est présent dans le dossier de l'utilisateur, il sera pris en considération pour constituer la politique de sécurité active.

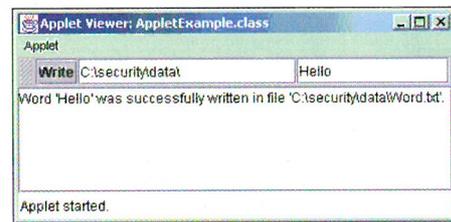
Donner des permissions à l'applet

Il est possible de rendre le bac à sable moins restrictif en lui ajoutant des permissions supplémentaires. Pour cela, on utilise l'utilitaire `policytool` (voir la [figure 3](#)) du JDK pour créer un fichier `.policy` décrivant les permissions supplémentaires à accorder. À noter, qu'il est également possible de créer un fichier `.policy` à l'aide d'un éditeur de texte.

Le fichier `AppletExample.policy` (voir le [listing 4](#)) donne à tout code (fichier `.class`) situé dans le dossier `C:\security\applet\`, la permission d'écrire le fichier `Word.txt` dans le dossier `C:\security\data\`.



> Figure 3 : l'utilitaire `policytool` du JDK.



> Figure 4 : exécution de `AppletExample2.bat`

Listing 4 : `AppletExample.policy`

```
grant codeBase "file:///C:/security/applet/" {
    permission java.io.FilePermission "C:\\security\\data\\Word.txt", "write";
};
```

Pour exécuter l'*applet* en tenant compte du fichier `AppletExample.policy`, lancer `AppletExample2.bat` (voir le [listing 5](#)). À noter qu'il aurait été également possible de créer un fichier `.java.policy` identique dans le répertoire d'utilisateur. Ce fichier aurait automatiquement été pris en compte sans préciser quoi que ce soit dans la ligne de commande, c'est-à-dire en utilisant `AppletExample1.bat`.

Listing 5 : `AppletExample2.bat`

```
appletviewer -J
-Djava.security.policy=AppletExample.policy
file:///C:/security/applet/AppletExample.html
```

En appuyant sur le bouton `Write`, tout se passe sans problème, et le fichier `Word.txt` est effectivement écrit dans le dossier `C:\security\data\`. Si on saisit un autre dossier et appuie sur le bouton `Write`, une exception `AccessControlException` est levée. Tout fonctionne comme prévu : ce qui est permis est possible, ce qui n'est pas permis, n'est pas possible.

Exemple d'une application

Nous allons utiliser une application en tout point similaire à l'*applet* précédente, à ceci près, qu'il s'agit maintenant d'une application et non plus d'une *applet*.

Tous les fichiers nécessaires au bon déroulement de l'exemple se trouvent dans le répertoire `C:\security\application\`.

Le fichier `ApplicationExample.java` contient le code de l'application qui a été compilée. L'application est livrée sous forme d'un fichier `jar` baptisé `ApplicationExample.jar`.

Nous allons procéder en 3 étapes et expliquer chaque fois ce qui se produit. Auparavant il faut lancer une fenêtre de lignes de commande et se placer dans le dossier `C:\security\application\`.

- Lancer `ApplicationExample1.bat` (voir le [listing 6](#)). Appuyer sur le bouton `Write`, tout se passe sans problème, le fichier est bien créé. Saisir un autre dossier, et appuyer de nouveau sur le bouton `Write`, tout se passe également sans problème, le fichier est bien créé.

Listing 6 : ApplicationExample1.bat

```
java -classpath ApplicationExample.jar ApplicationExample
```

En fait, dans le cas présent, aucun gestionnaire de sécurité n'est activé, l'application a donc le droit de faire tout ce qu'elle veut.

- Dans cette deuxième étape, nous allons activer un gestionnaire de sécurité. Lancer `ApplicationExample2.bat` (voir le [listing 7](#)). Appuyer sur le bouton `Write`, une exception `AccessControlException` est levée.

Listing 7 : ApplicationExample2.bat

```
java -Djava.security.manager  
-classpath ApplicationExample.jar ApplicationExample
```

Dans le cas présent, un gestionnaire de sécurité est activé. En l'absence d'indications supplémentaires, c'est la politique de sécurité par défaut qui est activée, c'est-à-dire le bac à sable.

- Dans cette troisième et dernière étape, nous allons activer un gestionnaire de sécurité et rendre le bac à sable moins restrictif par l'ajout d'une permission. Nous utiliserons pour cela le fichier `ApplicationExample.policy` (voir le [listing 8](#)). Le fichier `ApplicationExample.policy` donne à tout code situé dans le fichier jar `C:\security\application\ApplicationExample.jar`, la permission d'écrire le fichier `Word.txt` dans le dossier `C:\security\data\`.

Listing 8 : ApplicationExample.policy

```
grant codeBase "file:///C:/security/application/ApplicationExample.jar" {  
  permission java.io.FilePermission "C:\\security\\data\\Word.txt", "write";  
};
```

Lancer `ApplicationExample3.bat` (voir le [listing 9](#)), appuyer sur le bouton `Write`, tout se passe sans problème, et le fichier `Word.txt` est effectivement écrit dans le dossier.

Listing 9 : ApplicationExample3.bat

```
java -Djava.security.manager  
-Djava.security.policy=ApplicationExample.policy  
-classpath ApplicationExample.jar  
ApplicationExample
```

Applet et application

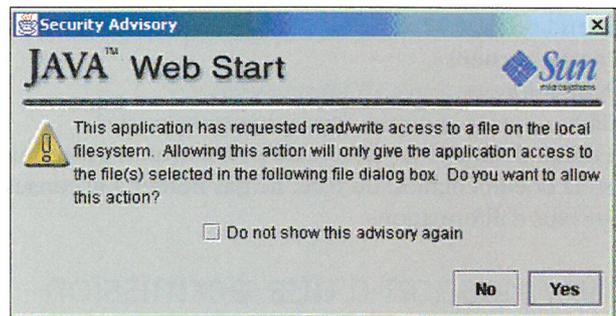
Dans le cas d'une *applet*, un gestionnaire d'applications est démarré d'office par le *Java Plugin* ou l'*Applet Viewer*. En revanche, dans le cas d'une application, il est nécessaire d'activer un gestionnaire de sécurité de manière volontaire.

Dans tous les cas de figure, que ce soit une application ou bien une *applet*, les mêmes mécanismes de sécurité sont utilisés, et la politique de sécurité est déterminée de la même manière.



Un mot sur Java Web Start

Idéalement, on devrait démarrer toute application Java téléchargée sous le contrôle d'un gestionnaire de sécurité. Dans la pratique, ce n'est pas forcément très évident, ni très pratique. Il existe pourtant une solution élégante et efficace offerte par le produit Java Web Start disponible récemment. Lorsqu'une application est téléchargée grâce à cette technologie, elle est automatiquement démarrée sous le contrôle d'un gestionnaire de sécurité. Ce dernier est un peu particulier. En effet, chaque fois que l'application requiert une permission, l'utilisateur est consulté pour savoir si oui ou non, il désire accorder cette permission (voir la [figure 5](#)). Chaque demande est accompagnée d'une petite explication claire qui montre bien les enjeux de la sécurité.

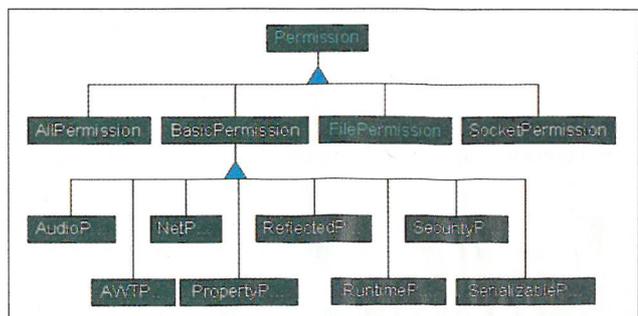


> Figure 5 : Java Web Start, exemple de demande de permission.

Java Web Start présente bien d'autres avantages, dont l'intérêt dépasse le cadre de cet article : mise à jour automatique de l'application, téléchargement automatique de la machine virtuelle adaptée, etc.

Permissions

Une permission est en fait une classe Java. Toute permission descend directement ou indirectement de la classe `Permission` qui se trouve dans le package `java.security`. La [figure 6](#)



> Figure 6 : hiérarchie des permissions.

montre la hiérarchie des permissions standard dans Java. Il est évidemment possible de créer de nouvelles permissions en héritant d'une permission existante, le plus souvent `Permission` et plus rarement `BasicPermission`.

L'accord d'une permission dans un fichier `.policy` se compose de 3 parties :

- le nom de la classe de permission qualifié par les packages, par exemple, `java.io.FilePermission`,
- la cible, c'est-à-dire ce qui est visé par la permission, par exemple, `C:\security\data\Word.txt`,
- les actions qui sont permises sur la cible, par exemple `write`, `read`. Dans le cas de permissions descendant de `BasicPermission`, cette partie ne doit pas être précisée.

Pour éclaircir ces notions, prenons un premier exemple complet :

- Permission : `java.net.SocketPermission`, permission relative à une socket ;
- Cible : `localhost:1024+`, tous les ports de la machine locale supérieurs à 1024 ;
- Actions : `accept`, `connect`, `listen`, les actions autorisées sur ces ports.

Prenons un second exemple. Il s'agit d'une permission qui descend de `BasicPermission`. Dans ce cas, on ne trouve pas de partie « actions ».

- Permission : `java.awt.AWTPermission` ;
- Cible : `accessClipboard`, l'accès au presse-papier.

Un détail de toutes les classes de permission est disponible dans la documentation du JDK : ne pas hésiter à le consulter pour plus d'informations.

Vérification d'une permission

Pour vérifier qu'une permission est accordée, les classes standard du JDK ont recours au gestionnaire de sécurité. Les classes utilisent pour cela un code qui s'apparente au code suivant, présenté ici à titre d'illustration (voir le [listing 10](#)) :

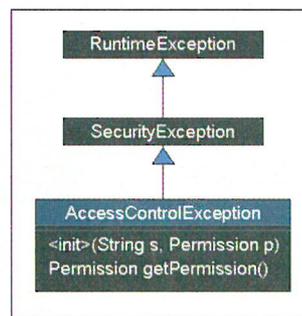
Listing 10 : code de vérification d'une permission

```
// import java.lang.SecurityManager;
import java.security.Permission;
import java.io.FilePermission;
SecurityManager sm =
    System.getSecurityManager();
if (sm != null) {
    Permission p = new FilePermission(
        "C:\\security\\data\\Word.txt", "write");

    sm.checkPermission(p);
}
```

Dans ce code, l'appel de la méthode `checkPermission` de la classe `SecurityManager` est le point précis où la permission est vérifiée. C'est cette méthode qui lève une exception `AccessControlException` si la permission n'est pas accordée.

L'exception `AccessControlException` descend indirectement de `RuntimeException`, c'est donc une exception non vérifiée (voir la [figure 7](#)).



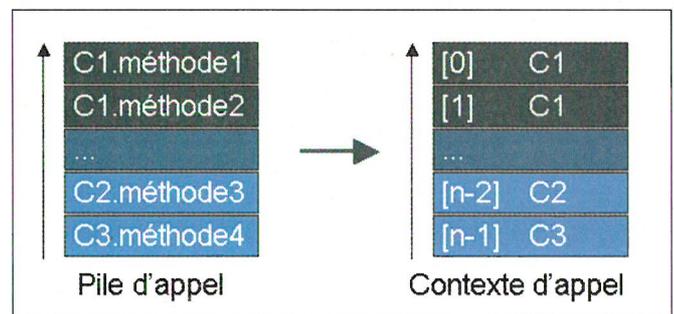
> Figure 7 : l'exception `AccessControlException`.

La méthode `getPermission` de l'exception `AccessControlException` est particulièrement intéressante, elle permet de connaître la permission non accordée à l'origine de l'exception.

Contexte d'appel

Mais comment une permission est-elle effectivement vérifiée ?

Que se passe-t-il dans la méthode `checkPermission` ? En fait, la vérification se fait à partir de ce que l'on appelle le contexte d'appel. Il est établi, à partir de la pile d'appels (voir la [figure 8](#)) de la manière qui suit. Pour chaque appel de méthode présent dans la pile d'appels, on ne conserve que la classe. La liste de classes prises à partir du sommet de la pile jusqu'à sa base constitue alors le contexte d'appel.



> Figure 8 : contexte d'appel.

Pour chaque classe de cette liste, il est alors vérifié si la classe a effectivement la permission.

Notons que les classes de la bibliothèque standard de Java ont toutes les permissions, ceci étant également valable pour les extensions. Dans la pratique, seul le code personnel est susceptible de ne pas avoir la permission.

Pour conclure

La sécurité Java 2 permet de contrôler finement l'exécution d'un `applet` ou d'une application et ainsi d'empêcher toute action hostile. Ceci n'est possible que parce que Java 2 repose sur une machine virtuelle.

La sécurité Java 2 est un sujet vaste et parfois ardu, nous n'avons ici fait qu'effleurer le sujet. Par exemple, nous n'avons pas ici couvert certaines notions telles que la signature de code. En effet, la sécurité Java 2 fournit des outils pour signer du code (plus exactement les `jar`). Le système de permissions permet alors d'accorder des permissions spécifiques à du code signé par tel ou tel signataire.

Enfin, la sécurité Java 2 est particulièrement extensible. Il est possible de créer son propre gestionnaire de sécurité et ses propres permissions. ■

```

{ final int i=10;
  class Inner
  {
    void f()
    { System.out.println(i);
    }
  }
  new Inner().f();
}

```

La méthode `f()` de la classe `Inner` de la méthode `MaClass.g()` peut manipuler la variable locale `i`. Celle-ci doit être déclarée `final`. Pourquoi ? En créant une tâche, on lui associe une pile qui ne sera utilisée que par elle. Il n'y a pas de conflit d'accès sur la pile. Si la méthode `f()` est appelée dans une autre tâche, il y aurait la possibilité d'avoir simultanément deux accès à la variable `i`. Comme il n'est pas possible de synchroniser l'accès aux variables de la pile, il faut interdire cela. Déclarer une variable `final` permet de garantir qu'elle ne sera jamais modifiée. Deux traitements peuvent consulter la variable, mais aucun ne peut la modifier. Que se passe-t-il si la méthode `g()` est terminée alors qu'une instance `Inner` est encore disponible ? Où la méthode `Inner.f()` va pouvoir trouver la variable `i` ?

```

class MaClass
{ Object g()
  { final int i=10;
    class Inner
    {
      void f()
      { System.out.println(i);
      }
    }
    return new Inner();
  }
}

```

Dans ce nouveau programme, la fonction `g()` retourne une instance `Inner`, interne à la fonction. L'appelant de `g()` peut utiliser l'instance interne après l'exécution de `g()`.

Tous les variables locales utilisées sont recopiées lors de l'appel du constructeur de l'instance. La classe `Inner` est convertie par le compilateur en classe classique comme ceci :

```

class MaClass$1
{ private final int val$i;
  MaClass$1(int val$i)
  { this.val$i =val$i;
  }
  void f()
  { System.out.println(val$i);
  }
}

```

Ne devant pas être confondue avec une classe interne de même nom, le compilateur déclare la classe `$1`, mais elle n'appartient pas à une fonction. Les classes de fonction ne sont visibles que pour chaque fonction. La classe externe est traduite ainsi :

```

class MaClass.
{

```

```

Object g()
{ final int i=10;
  return new Inner(i);
}

```

La classe interne ajoutée dans les constructeurs, toutes les variables `final` nécessaires. Les instances internes possèdent une copie des variables locales de la méthode. Il n'y a pas de risque d'incohérence entre les deux copies car elles ne peuvent pas être modifiées. Si une variable locale à la méthode n'est utilisée que par le constructeur, le compilateur le détecte et ne génère pas d'attribut pour celle-ci. En effet, cette variable n'est plus nécessaire pour les autres méthodes.

Les classes anonymes

Java propose également la notion de classe anonyme. Une classe anonyme est une classe déclarée là où une variable est nécessaire.

```

class MaClass
{ void f()
  { class MyThread implements Runnable
    { public void run()
      { for (int i=0;i<10;++i)
        System.out.println("Hello");
      }
    };
    new MyThread().start();
  }
}

```

La classe `MyThread` ne sert qu'une seule fois. Son nom importe peu. Pour éviter de devoir déclarer des classes techniques avec un nom, les classes anonymes permettent de simplifier ce code.

```

class MaClass
{ void f()
  { new Thread(new Runnable()
    { public void run()
      { for (int i=0;i<10;++i)
        System.out.println("Hello");
      }
    })
    .start();
  }
}

```

L'appel du constructeur de `MyThread()` est remplacé par une déclaration spéciale de la classe. Ce code est converti par le compilateur comme ceci :

```

class MaClass
{ void f()
  { class $1 implements Runnable
    { public void run()
      { for (int i=0;i<10;++i)
        System.out.println("Hello");
      }
    }
  };
  new $1().start();
}

```

Puis comme ceci :

```
class MaClass$1 implements Runnable
{
    public void run()
    { for (int i=0;i<10;++i)
      System.out.println("Hello");
    }
}
class MaClass
{ void f()
  { new Thread(new MaClass$1()).start();
  }
}
```

Une classe anonyme peut avoir des initialisations d'attributs mais ne peut pas avoir de constructeur.

```
new Runnable()
{ String msg="hello";
  String upper=msg.toUpperCase();
  ...
}.start();
```

Si la classe anonyme utilise des variables locales de la fonction `f()`, un constructeur est ajouté pour initialiser les variables privées de l'instance. Il faut déclarer les variables de la méthode, utilisées par la classe anonyme, en `final`. Dans l'exemple suivant, la variable privée se nomme `max`.

```
class MaClass
{ void f(final int max)
  { new Thread(new Runnable()
    { public void run()
      { for (int i=0;i<max;++i)
        System.out.println("Hello");
      }
    }).start();
  }
}
```

Contrairement aux exceptions, cela ne fait pas partie de la signature de la méthode. Une sous-classe n'est pas obligée d'avoir le paramètre `max` en `final`.

```
class MaClassEx extends MaClass
{ void f(int max)
  {
  }
}
```

La méthode `MaClassEx.f(int max)` surcharge la méthode `MaClass.f(final int max)`. L'inverse est également vrai.

Le code de `MaClass` devient :

```
class MaClass$1 implements Runnable
{
    final int val$max;
    MaClass$1(int val$max)
    { this.val$max=val$max;
    }
    public void run()
    { for (int i=0;i<val$max;++i)
      System.out.println("Hello");
    }
}
```

```

}
}
class MaClass
{ void f(final int max)
  { new Thread(new MaClass$1(max)).start();
  }
}
```

Il faut déclarer le paramètre `max` comme `final` car il est utilisé dans une méthode de la classe anonyme. L'instance anonyme peut exister après l'exécution de la méthode `f()`.

Le traitement est identique avec les classes internes non anonymes. Tous les variables locales à la méthode sont recopiées par le constructeur de l'instance anonyme. Une classe anonyme peut être déclarée en dehors d'une méthode. Pour cela, il faut utiliser la syntaxe permettant d'initialiser les attributs.

```
class MaClass
{ Object o=new Object()
  { bool f()
    { return o==null;
    }
  };
}
```

La visibilité dépend du contexte de création. Si la classe anonyme est déclarée lors de l'initialisation d'un attribut, elle peut consulter tous les attributs de l'instance externe. Si la classe anonyme est déclarée lors de l'initialisation d'un attribut statique, elle ne peut consulter que les attributs statiques.

```
class MaClass
{ static Object o=new Object()
  { bool f()
    { return o==null;
    }
  };
}
```

La classe anonyme est dans ce cas une `inner classe` anonyme statique.

```
class MaClass
{ static class $1
  { bool f()
    { return o==null;
    }
  };
  static Object o=new $1();
}
```

Le code final ressemble alors à ceci :

```
static class MaClass$1
{ bool f()
  { return o==null;
  }
};
class MaClass
{
  static Object o=new MaClass$1();
}
```

Dans un prochain article, nous continuerons de faire connaissance avec nos drôles de composants. ■

Le grand défilé de collections de VB.Net



VB6 a changé ses vieilles collections pour des beaux habits neufs. Ça fait tout de suite plus «net». Richard Clark – rc@c2i.fr

Avant d'attaquer le cœur de cet article, c'est-à-dire les collections dans Microsoft.NET Framework, nous allons vous présenter quelques nouvelles fonctionnalités de VB.Net pour bien comprendre les exemples de code.

Constructeurs

Une autre fonctionnalité très attendue de Visual Basic.NET (notamment par moi!), c'est la possibilité de créer des constructeurs.

Imaginez une classe `cEmploye`. Celle-ci possède deux propriétés en lecture-écriture :

- `NombreDeMois` ;
- `SalaireAnnuel` ;

Et une propriété en lecture seule :

- `SalaireMensuel`.

Cette dernière est le résultat d'un calcul : `SalaireAnnuel / NombreDeMois`. (Je m'excuse auprès des puristes de ne pas avoir écrit la propriété `NombreDeMois` comme il faut, *i.e.* avec «Public Property NombreDeMois, etc.»). Son code est le suivant :

```
Public Class cEmploye
    Private mvarSalaireAnnuel As Integer ' copie locale
    Public NombreDeMois As Integer ' copie locale
    '-----LES PROPRIETES-----
    Public Property SalaireAnnuel() As Integer
        Get
            SalaireAnnuel = mvarSalaireAnnuel
        End Get

        Set
            mvarSalaireAnnuel = Value
        End Set
    End Property
    '-----LA PROPRIETE EN LECTURE SEULE-----
    Public ReadOnly Property SalaireMensuel() As Single
        Get
            SalaireMensuel = SalaireAnnuel / NombreDeMois
        End Get
    End Property
End Class
```

Jusqu'à présent, pour pouvoir utiliser correctement cette classe, il fallait à tout prix affecter une valeur au `NombreDeMois` pour pouvoir connaître le `SalaireMensuel`. Cela signifiait que l'utilisateur de votre classe devait à tout prix savoir comment utiliser votre classe. S'il oubliait d'affecter une valeur à `NombreDeMois`, son application plantait et il vous insultait au téléphone (ou vous jetait sa pizza à la figure...)

Vous pouvez maintenant définir des constructeurs qui empêcheront ce phénomène du vol de pizza dans les couloirs. Pour cela, redéfinissez le constructeur `New` de votre classe :

```
Public Sub New(ByVal iNombreDeMois As Integer)
    NombreDeMois = iNombreDeMois
End Sub
```

Pour pouvoir créer une instance de votre classe, votre utilisateur sera obligé d'écrire le code suivant :

```
Dim objEmploye As New cEmploye(13)
ou
Dim objEmploye As cEmploye
objEmploye = New cEmploye(13)
```

Surcharge

Toute méthode d'un objet peut être surchargée. Cela signifie qu'une même méthode peut être définie de plusieurs façons. Regardez le code suivant d'une classe `cPerson` :

```
Public Class cPerson
    Private msNom As String
    Private msPrenom As String

    Public Property Nom() As String
        Get
            Nom = msNom
        End Get

        Set
            msNom = Value
        End Set
    End Property

    Public Property Prenom() As String
    ...
```

```

End Property

Public Overloads Sub New(ByVal sNom As String)
    Nom = sNom
End Sub

Public Overloads Sub New(ByVal sNom As String, ByVal sPrenom As String)
    Nom = sNom
    Prenom = sPrenom
End Sub

End Class

```

La méthode pour créer un objet `cPerson` est définie de deux façons différentes (redéfinitions du constructeur `New`) : soit on ne donne que le nom de la personne, soit on donne le nom et le prénom. La création d'un objet `cPerson` pourra donc se faire ainsi :

```

Dim objToto As cPerson
objToto = New cPerson("Clark")
objToto.Prenom = "Richard"
ou
Dim objToto As cPerson
objToto = New cPerson("Clark", "Richard")
ou
Dim objToto = New cPerson("Clark", "Richard")

```

Dans le cas d'une propriété `Item` d'une classe `Collection`, on peut maintenant la définir avec l'index ou la clé sans utiliser de variant. Ainsi, VB n'aura pas besoin « d'évaluer » le type de la variable. Les performances seront donc accrues.

Les collections de Microsoft.NET Framework

Si, comme moi, vous aviez l'habitude d'utiliser des objets type collection avec Visual Basic 6, sachez que MS.NET met à votre disposition de nouveaux types de collections. Vous pouvez toujours utiliser les anciennes collections qui sont dans le *Namespace Microsoft.VisualBasic.Compatibility.VB6*.

```
Private mCol As Microsoft.VisualBasic.Compatibility.VB6.Collection
```

Ce premier article vous montre comment implémenter cet objet. Le but va être de créer une classe collection `cEmployes` d'objets `cEmploye`.

Créons `cEmploye`

Cette classe est simple car le sujet n'est pas là :

```

Public Class cEmploye

    Private m_sNom As String

    Public Property Nom() As String
        Get
            Nom = m_sNom
        End Get
        Set
            m_sNom = Value
        End Set
    End Property

```

```

Public Sub New(ByVal sNom As String)
    Nom = sNom
End Sub

End Class

```

Remarquez que nous avons redéfini le constructeur. On impose ainsi qu'une classe `cEmploye` ne peut être créée que si l'on donne un nom à notre employé :

```
Dim objE = New cEmploye("Richard Clark")
```

Collections VB6

Nous arrivons au cœur du sujet. Dans un premier temps, on va ajouter une variable privée à notre classe `cEmployes` :

```

Public Class cEmployes
    Private mCol As Microsoft.VisualBasic.Compatibility.VB6.Collection

    Public Sub New()
        mcol = New Microsoft.VisualBasic.Compatibility.VB6.Collection()
    End Sub
    ...
End Class

```

Pour implémenter l'ajout d'un employé :

```

Public Function Add(ByVal sNom As String, ByVal sKey As String) As cEmploye
    Dim objE As New cEmploye(sNom)
    mcol.Add(objE, sKey)
    Add = objE
End Function

```

Vous voyez que par défaut, j'impose l'utilisation d'une clé (mais c'est ma propre façon de coder).

Pour les autres méthodes et propriétés (`Item`, `Count` et `Remove`), rien ne change par rapport à VB6. Je n'irai donc pas plus loin.

Reste maintenant à permettre au consommateur de notre classe de pouvoir faire des itérations. Sous VB6, cela tenait plus de la recette de cuisine que d'autre chose (rappelez-vous le `IUnknown` de COM+, l'ID à -4, etc.)

C'est maintenant beaucoup plus simple, il suffit d'utiliser l'interface `IEnumerator` mise à disposition par l'objet `Collection` :

```

Public Function GetEnumerator() As IEnumerator
    GetEnumerator = mcol.GetEnumerator()
End Function

```

C'est tout ! On peut faire maintenant de l'itération sur notre objet :

```

Dim c As New cEmployes()
c.Add("Richard Clark", "clé")
For Each objTemp In c
    Debug.WriteLine(objTemp.Nom)
Next

```

ArrayList

Cette classe stocke des objets dans un tableau dynamique (sans clé). Notre classe `cEmploye` est la même. En revanche, la classe `cEmployes` a quelque peu changé. En voici le code :

```
Public Class cEmployes
Private mCol As ArrayList

Public Sub New()
    mCol = New ArrayList()
End Sub

Public Function GetEnumerator() As IEnumerator
    GetEnumerator = mCol.GetEnumerator
End Function

Public Function Add(ByVal sNom As String) As cEmploye
    Dim objE As New cEmploye(sNom)
    mCol.Add(objE)
    Add = objE
End Function

Public ReadOnly Property Item(ByVal iIndex As Integer) As cEmploye
    Get
        Item = mCol.Item(iIndex)
    End Get
End Property

Public ReadOnly Property Count() As Integer
    Get
        Count = mCol.Count
    End Get
End Property

Public Sub Remove(ByVal sKey As String)
    mCol.Remove(sKey)
End Sub

End Class
```

Le principe est donc identique. En revanche, pour obtenir un élément dans la collection, il faut connaître son index.

HashTable

Cette collection associe à chaque élément de la collection une clé obligatoire. Le code est similaire sauf pour la déclaration :

```
Private mHashCol As Hashtable
```

Il en est de même pour l'énumération et l'ajout d'un élément qui se font sur le tableau des valeurs :

```
Public Function GetEnumerator() As IEnumerator
    GetEnumerator = mHashCol.Values.GetEnumerator
End Function

Public Function Add(ByVal sNom As String, ByVal sKey As String) As cEmploye
    Dim objE As New cEmploye(sNom)
    mHashCol.Add(sKey, objE)
    Add = objE
End Function
```

SortedList

Cette classe stocke des objets dans un tableau dynamique avec une clé et trie les objets suivant la clé.

Elle est plus lente qu'une collection de type *HashTable*, mais offre cette fonctionnalité de tri en plus et permet également d'accéder à un élément de la collection par sa clé ou par son index. Le code est similaire à celui d'une *HashTable*. En revanche, comme un élément peut s'obtenir soit par son index, soit par sa clé, on va surcharger cette propriété :

```
Public Overloads ReadOnly Property Item(ByVal iIndex As Integer) As cEmploye
    Get
        Item = mCol.GetByIndex(iIndex)
    End Get
End Property
Public Overloads ReadOnly Property Item(ByVal sKey As String) As cEmploye
    Get
        Item = mCol.Item(sKey)
    End Get
End Property
```

L'exemple ci-dessous montre comment une *SortedList* peut être utilisée :

Code	Résultat
Dim c2i As New cEmployes()	élément 4
c2i.Add(«élément 1», «C»)	élément 3
c2i.Add(«élément 2», «D»)	élément 1
c2i.Add(«élément 3», «B»)	élément 2
c2i.Add(«élément 4», «A»)	
Dim objTemp As cEmploye	élément 1
For Each objTemp In c2i	élément 2
Debug.WriteLine(objTemp.Nom)	
Next	
Debug.WriteLine(c2i.Item(2).Nom)	
Debug.WriteLine(c2i.Item(«D»).Nom)	

La collection se charge du tri. Intéressant non ?

Queue

Différente de toutes celles présentées jusqu'à présent, cette collection repose sur la technique du *First In-First Out* (premier entré, premier sorti). Jusqu'à présent, faire une collection de ce type, rapide, avec Visual Basic était assez difficile (voire impossible si la rapidité était notre souci premier).

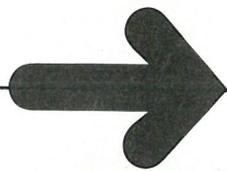
Vous y avez maintenant accès en natif !

L'implémentation d'une telle collection est très simple (je passe sur les constructeurs et méthodes usuelles) :

```
Public Class cEmployes
...
Public Function EnQueue(ByVal sNom As String) As cEmploye
    Dim objE As New cEmploye(sNom)
    mCol.Enqueue(objE)
    EnQueue = objE
End Function

Public Function DeQueue() As cEmploye
    DeQueue = mCol.Dequeue()
End Function

Public ReadOnly Property Peek()
    Get
        Peek = mCol.Peek()
    End Get
End Property
End Class
```



SOMMAIRE CD

Numéro 29 • Février 2001

> Pour utiliser le CD-ROM, cliquez sur le fichier index.htm figurant sur la racine du CD. Chaque logiciel est accompagné d'un texte de présentation, une image ainsi qu'un bouton de lancement ou d'installation. Cliquez sur l'un de ces boutons pour installer ou lancer le programme.



Sur votre CD-Rom



PROGRAMMES

• .Net Mobile de Microsoft

(version bêta, en anglais).

L'informatique du futur sera communicante ou ne sera pas! À cette fin, Microsoft, à travers .NET vous propose ce SDK pour construire des applications web mobiles pour téléphones portables et Pocket PC.

• Universal Description Discovery and Integration (UDDI) SDK de Microsoft

(version bêta, en anglais, pour Windows 2000).

Ajoutez des fonctionnalités de services web à vos outils de développement !

Ce SDK fournira au programmeur VB, les outils nécessaires pour travailler avec le registre UDDI (*Universal Description, Discovery and Integration*) permettant à n'importe quelle application de communiquer avec une autre *via* Internet.

• Rational Rose 2001

(version complète, 30 jours, en anglais).

La solution leader de modélisation visuelle et de développement à base de composants et du *round-trip engineering* pour les environnements majeurs du marché.

• Dreamweaver 4.0 de Macromedia

(version complète, 30 jours, en anglais).

L'outil professionnel de conception et de réalisation de sites web.

• JRun 3.0 Studio d'Allaire

(version complète, 30 jours, en anglais, après enregistrement sur le Web).

Environnement de développement intégré simple d'utilisation pour la création et le déploiement d'applications Java.

• WebTrends Professional Suite

(version complète, 30 jours, en anglais).

Solution complète permettant aux gestionnaires de sites intranet et Internet d'effectuer des analyses et de produire des rapports, afin d'évaluer les performances, l'efficacité, et la qualité de navigation.

BOÎTE À OUTILS (VERSIONS D'ÉVALUATION)

- Mozilla Seamonkey 0.7.
- Kernel 2.4 for Linux.
- PHP 4.0.4.
- Active Maker v.1.0 for Visual Basic.
- Multimedia Fusion 1.2.
- NetObjects Fusion 5.0.
- Windows Help Designer (HTML Edition).

SHAREWARES

- ChaosFx N.C • CinePlayer Editor 1.45 • Feurio 1.52
- Searchopia 32 3.00 • Sisoft Sandra 2000 6.49 • Anfy 3D 1.43
- CensorX 1.0 • Dutch's HTML Editor V2.1 3.0 • IntelLogo 1.2
- Plan B HelpMaker 1.0 • Gator Edit 32 2.09 • Mail Assistant 1.4
- Reactor 3.01 • SimpleHTML 2.2.1 • SOSCrash NC • RTScope ActiveX Control 3.0 • TVMenuBar 1.0196 • Ultra edit 32 8.0.

LA COMPILATION S.T.R.

Les meilleurs outils du marché pour mettre à jour votre boîte à outils.

RÉPERTOIRE DE LA RÉDACTION :

Retrouvez les sources des articles dans le dossier REDAC à partir de la racine du CD. Vous y trouverez notamment, les sources des articles du n°29 de *Programmez !*.

Programmez ! Le 1^{er} magazine de tous les langages • Supplément interactif N°29

Rational Rose 2001

La solution leader de modélisation visuelle et de développement à base de composants et de *round-trip engineering* pour les environnements majeurs du marché.
(version COMPLETE, 15 jours, en anglais, après enregistrement sur le Web)

OUTILS MICROSOFT

Universal Description Discovery and Integration (UDDI)

Ajoutez des fonctionnalités de services Web à vos outils de développement
(version bêta, en anglais)

.Net Mobile

Construisez des applications Web mobiles pour téléphones portables et Pocket PC !
(version bêta, en anglais)

Dreamweaver 4.0

de Macromedia
L'outil professionnel de conception et de réalisation de sites Web.
(version COMPLETE, 30 j., en anglais)

JRun 3.0 Studio d'Allaire

Environnement de développement pour la création et le déploiement d'applications Java.
(version COMPLETE, 30 j., en anglais, après enregistrement sur le Web)

WebTrends Professional Suite

Solution complète permettant d'évaluer les performances, l'efficacité et la qualité de navigation.
(version COMPLETE, 30 j., en anglais)

Boîte à outils : Mozilla 0.7 • Kernel 2.4 for Linux • Php 4
• Active Maker Lite • Multimedia Fusion 1.2 • NetObjects Fusion 5.0 • Windows Help Designer.

Quand vous ajoutez un élément (**EnQueue**), il se place automatiquement à la fin de votre collection. Pour obtenir le premier élément de la collection (donc le premier entré), vous utilisez la méthode **DeQueue** qui en plus, le retire de la collection. La méthode **Peek**, en revanche, vous donne le premier élément sans le retirer de la collection. Ainsi, l'on a avec le code test suivant :

Code	Résultat
Dim c2i As New cEmployes()	4
c2i.Enqueue(«élément 1»)	élément 1
c2i.Enqueue(«élément 2»)	3
c2i.Enqueue(«élément 3»)	élément 2
c2i.Enqueue(«élément 4»)	3

```

Dim objTemp As cEmploye
Debug.WriteLine(c2i.Count)
objTemp = c2i.DeQueue
Debug.WriteLine(objTemp.Nom)
Debug.WriteLine(c2i.Count)
objTemp = c2i.Peek
Debug.WriteLine(objTemp.Nom)
Debug.WriteLine(c2i.Count)
    
```

Vous avez aussi la possibilité avec cette collection de faire des itérations avec **GetEnumerator**.

Stack

Le principe de **Stack** est similaire à celui de la collection **Queue** sauf que le dernier élément entré, est le premier sorti !

```

Public Class cEmployes
...
Public Function Push(ByVal sNom As String) As cEmploye
    Dim objE As cEmploye
    objE = New cEmploye(sNom)
    mCol.Push(objE)
    Push = objE
End Function

Public Function Pop() As cEmploye
    Pop = mcol.Pop
End Function

Public ReadOnly Property Peek()
    Get
        Peek = mcol.Peek
    End Get
End Property

End Class
    
```

Push place un élément dans la collection, **Pop** retire et renvoie le premier élément de la collection, c'est-à-dire le dernier entré et **Peek** retourne le premier élément sans le retirer de la collection. Ainsi, l'on a avec le code test suivant :

Code	Résultat
Dim c2i As New cEmployes()	4
c2i.Push(«élément 1»)	élément 4
c2i.Push(«élément 2»)	3
c2i.Push(«élément 3»)	élément 3
c2i.Push(«élément 4»)	3

```

Dim objTemp As cEmploye
Debug.WriteLine(c2i.Count)
objTemp = c2i.Pop
    
```

```

Debug.WriteLine(objTemp.Nom)
Debug.WriteLine(c2i.Count)
objTemp = c2i.Peek
Debug.WriteLine(objTemp.Nom)
Debug.WriteLine(c2i.Count)
    
```

Performances

Qu'en est-il des performances des différentes collections proposées par MS.NET ?

Avertissement : les tests ont été effectués avec la bêta 1 de MS.NET, il faudra donc les revoir lors de la version définitive.

La comparaison concerne les collections suivantes :

VB6.Collection, **ArrayList**, **HashTable** et **SortedList**.

Bien entendu, chacune de ces collections a ses propres avantages liés au contexte d'utilisation. Il ne s'agit pas de dire ici, celle-ci est mieux que celle-là.

Le test comporte trois parties :

- remplissage de la collection avec 10 000 objets,
- recherche aléatoire d'un objet dans la collection et ce, 5 000 fois de suite,
- itération sur l'ensemble des éléments de la collection.

Vous trouverez sur le site de *Programmez!* le code source de ce test.

On obtient le résultat suivant sur un PII 400Mhz, Windows 2000 Server, 256 Mo de RAM :

Attention, il ne faut prendre que les ordres de grandeur de ces chiffres (16 ms est équivalent à 0 ms, prenez un delta d'erreur de 50 ms).

Les résultats sont ceux que l'on pouvait espérer :

VB6 Collection	ArrayList	HashTable	SortedList
Remplissage : 203	Remplissage : 46	Remplissage : 203	Remplissage : 562
Recherche : 360	Recherche : 266	Recherche : 313	Recherche : 500
Itération : 62	Itération : 16	Itération : 0	Itération : 0
Total : 625	Total : 328	Total : 516	Total : 1062

Remplissage

ArrayList est le plus performant puisqu'il ne fait rien de spécial pendant son remplissage.

HashTable, lui, doit lier la clé à l'index, d'où un temps plus important.

SortedList doit lier la clé à l'index et effectuer le tri. Il est donc normal qu'il soit le plus lent.

VB6 collection se situe globalement au niveau de la **HashTable**.

Recherche

Pour **VB6 Collection**, **ArrayList**, **HashTable**, on arrive aux mêmes performances. Une fois de plus, **SortedList** est le plus pénalisé. Je ne vois pas pourquoi et je pense que ce problème devrait être amélioré avec la version définitive.

Recherche aléatoire
Remplissage : 921
Recherche : 1234
Itération : 47
Total : 2202

Itération

VB6 Collection est légèrement en retrait par rapport aux autres. On notera toutefois que dans tous les cas, le temps est proche de zéro.

Comparaison avec VB6

Un même style de test à été effectué dans les mêmes conditions sous VB6 (code compilé), uniquement avec la classe `Collection`, bien sûr. Les résultats parlent d'eux-mêmes : Même l'implémentation de VB6 Collection dans VB.NET est beaucoup plus rapide que son homologue sous VB6 (22 secondes contre 6 secondes soit presque 4 fois plus rapide!).

Conclusion

MS.NET nous propose de nombreuses collections différentes avec pour chacune, une utilisation spécifique (il en existe encore d'autres que nous n'avons pas présentées ici comme `BitArray`, `StringCollection`, etc.). Mais dans tous les cas, les performances sont au rendez-vous. Si vous souhaitez conserver une collection du même type que celle que vous utilisiez sous VB6, prenez la `HashTable`. Une collection sans clé, `ArrayList` s'impose par sa rapidité. Bref, les choix sont multiples et vont dépendre de vos applications.

Petit aparté sur le nouvel objet Debug

Du temps de Visual Basic 4, l'objet `Debug` ne possédait qu'une seule méthode :

```
Debug.Print sMessage
```

Puis est apparue avec VB5 la méthode `Debug.Assert` qui permettait de créer un arrêt conditionnel. Autant le dire, cette version était sympa, mais sans plus...

Maintenant, avec VB.NET, vous avez un véritable objet avec de multiples fonctionnalités.

Premièrement, vous pouvez écrire classiquement dans la fenêtre `OutPut` de l'environnement de développement, mais vous pouvez également ajouter vos propres `Listeners` (on y reviendra dans un autre article).

Écrire dans la fenêtre OutPut.

Pour écrire dans la fenêtre `OutPut`, vous avez maintenant plusieurs méthodes :

```
Debug.Write(sMessage)
Debug.WriteLine(sMessage)
```

Dans ces deux cas, vous écrivez systématiquement dans la fenêtre `OutPut` la valeur de `sMessage`. Dans le premier cas, il n'y a pas de retour de ligne et les textes apparaissent les uns derrière les autres. Plus intéressant, vous pouvez choisir d'écrire dans la fenêtre `OutPut` si une condition est vérifiée :

```
Debug.WriteLineIf(test boolean, sMessage)
Debug.WriteLineIf(test boolean, sMessage)
```

Cela vous permet, sans arrêter le programme (comme le fait `Assert`), d'être informé quand une condition est vérifiée. Vous avez la possibilité de définir l'indentation de votre sortie, par exemple :

Code	Résultat
Debug.WriteLine("Coucou")	Coucou
Debug.Indent = 5	
Debug.WriteLine("Coucou1")	coucou1
Debug.Indent = 10	
Debug.WriteLine("Coucou2")	Coucou2
Debug.Indent = 0	
Debug.WriteLine("Coucou3")	Coucou3

Vous pouvez même définir la taille de l'indentation avec la propriété `Debug.IndentSize`.

Une erreur fatale

Si sur un point particulier vous souhaitez être informé de tout ce qui se passe, vous pouvez utiliser la méthode `Fail` qui s'utilise des deux façons suivantes :

```
Debug.Fail("Aie, y'a un problème")
Debug.Fail("Aie, y'a un problème", "Faudrait peut-être faire quelque chose")
```

Dans ce cas là, Visual Basic affiche une fenêtre contenant votre (ou vos 2) message(s) avec en plus des informations complémentaires sur l'endroit où cela s'est produit, etc. Vous avez alors le choix classique : `Abort`, `Retry`, `Ignore`. Mettre le programme en pause.

`Debug` possède toujours la méthode `Assert` qui a été, elle aussi, améliorée. Elle possède maintenant plusieurs constructeurs :

```
Debug.Assert bCondition
Debug.Assert bCondition, sMessage
Debug.Assert bCondition, sMessage, sMessageDetaillé
```

Dans ces trois cas, la fenêtre générée par `Fail` apparaît (`Abort`, `Retry`, `Ignore`). Dans le cas de la fenêtre `OutPut`, c'est exactement le même fonctionnement que la méthode `Fail`.

Les Listeners

Vous pouvez maintenant choisir de ne plus écrire dans la fenêtre `OutPut` mais dans un ou des fichiers texte. Pour cela, il vous faut ajouter un ou des `listeners`.

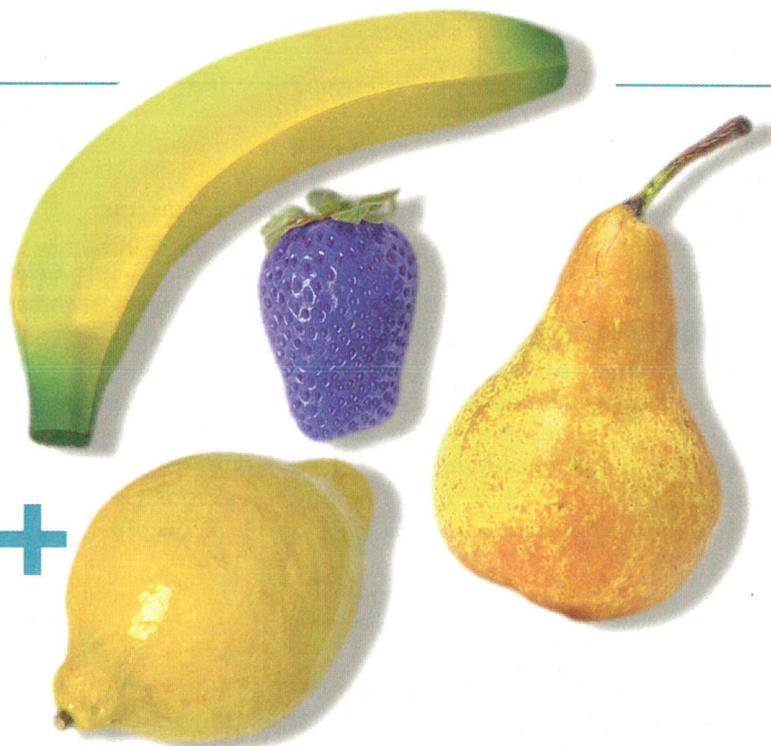
`Debug` possède une propriété collection `Listeners` (objet `TraceListeners`) de `TraceListener`. Chaque `TraceListener` vous permet d'écrire dans un fichier texte.

Nous reviendrons plus tard sur cette fonctionnalité. ■

L'auteur :

Richard Clark – rc@c2i.fr – <http://www.c2i.fr>
Le portail des développeurs francophones
VB – VB.NET – ASP – ASP.NET

Les petits « plus » de C++



Nous allons commencer une nouvelle série d'articles sur le C++. Sous forme de jeu, nous allons nous intéresser à des détails subtils de ce langage. Un petit exemple vous est proposé et *vous devez trouver l'erreur insérée*. Ri-go-lo, non ?

L'auteur :

Philippe Prados est ingénieur expert chez IBM Global Services, département Nouvelles technologies.

<http://www.prados-obj.nom.fr>

http://www.software.ibm.com/ad/visualage_c++/

<http://egcs.cygnum.com/>

Mais que serait le résultat sans la résolution du problème ! Des explications concernant les erreurs insérées dans l'exemple vous sont données, avec éventuellement, différentes techniques pour les contourner. Chaque exemple permet d'introduire une règle de développement afin d'éviter que l'erreur ne se représente. Les règles de qualité se déduisent de l'expérience.

L'objectif est de faire connaître les sources d'erreur les plus subtiles qui risquent d'exister dans tous les développements C++. Certains problèmes sont faciles, d'autres beaucoup moins. Tout programmeur C++ devrait être capable de répondre correctement à la majorité des problèmes exposés. Lors des développements, il faut bien prendre conscience des risques évoqués. Ceci permet d'écrire un programme sans trop d'erreurs, et surtout, de ne pas perdre de temps dans la localisation de celles-ci.

Les exercices sont réduits à leur plus simple expression. Aucun `#include` n'est indiqué pour ne pas alourdir les sources. Toutes les allocations « mémoire » sont considérées

réussies. Il n'existe pas de réponse du type : « L'allocation n'est pas vérifiée, il est donc possible d'écrire à l'adresse NULL ». Tous les problèmes doivent être portables et ne pas dépendre d'une version particulière du compilateur. Certains d'entre eux fonctionnent avec un compilateur, mais pas avec un autre. Il faut, dans ce cas, que vous indiquiez pourquoi.

La premier exemple est très simple.

Question 1

```
short add(short i, short j)
{
    return i+j;
}
```

Le compilateur peut annoncer un *warning*, pourquoi ?

Si "`sizeof(short) == sizeof(int)`", il n'y a pas de problème, le compilateur ne signale pas d'erreur.

Par contre, si "`sizeof(short) < sizeof(int)`", un message est affiché. En effet, toutes les opérations sont faites en `int`. Les variables `i` et `j` sont dans un premier temps converties en entiers. L'addition est ensuite calculée, le résultat de celle-ci est un entier. Ensuite, le compilateur cherche à convertir cet entier en `short`.

```
{
    return (short)((int)i+(int)j)
}
```

Il y a perte de précision. La donnée résultante peut être erronée, ce qui est signalé par un *warning*. Cela indique qu'il

est inutile d'utiliser le type `short` pour optimiser un programme, mais que celui-ci n'est utile que pour réduire la taille mémoire du logiciel. Le compilateur passe beaucoup de temps à convertir les `short` en `int` et inversement, ce qui ralentit le programme. L'arithmétique `short` n'est pas présente dans les compilateurs. On peut le regretter. De plus, le type `short` peut entraîner un problème de portabilité lors de la résolution des méthodes surchargées.

```
void f(int); // f1
void f(unsigned int); // f2

unsigned short s;
f(s);
```

Si les types `short` et `int` sont de taille identique, l'appel de `f(s)` sera résolu sur l'appel de la version `f2`. Sinon, le type `int` est capable d'avoir toutes les valeurs possibles de `unsigned short`, alors la promotion de type, convertit le `unsigned short` en `int`, et c'est l'appel de `f1` qui est généré. Ne pas utiliser `short` à la place de `int` si cela n'est pas utile. La deuxième question est un peu plus complexe.

Question 2

```
class CString
{
public:
    CString(const char* src)
    {
        pt=new char[strlen(src)+1];
        strcpy(pt,src);
    }
    ~CString()
    {
        delete [] pt;
    }
};

CString f()
{
    CString str("test");
    return str;
}

void main()
{
    CString x=f();
}
```

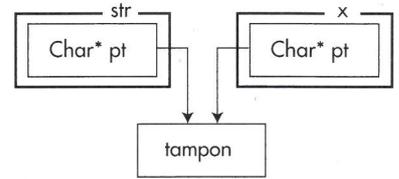
C'est incorrect, pourquoi ?
 Pour bien comprendre l'erreur, il faut tout d'abord savoir comment le compilateur permet à une fonction de renvoyer un objet. La fonction `f()` reçoit un pointeur caché indiquant où l'objet retourné doit être créé. L'appel de la fonction `f()` se traduit comme cela :

```
void f(CString* _ret) // ret est un pointeur caché
{
    CString _tmp("test");
    *_ret=CString::_CString(_tmp); // Appel du ctr de copie
```

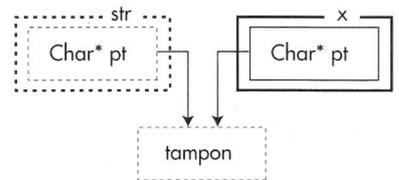
```
_tmp::~~CString(); // Destruction de l'obj _tmp
}

void main()
{
    CString x; // Déclaration de l'objet // sans construction
    f(&x); // Appel de la fonction
}
```

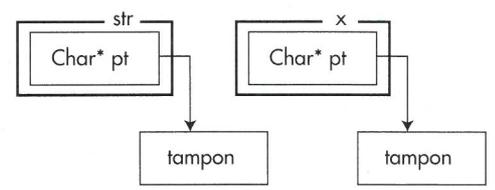
L'instruction `return` est traduite par l'appel du constructeur de copie sur le pointeur caché. La classe `CString` n'en possède pas. Le compilateur utilise alors le constructeur de copie par défaut. Celui-ci effectue une copie binaire membre à membre des éléments de la structure.



Le pointeur `pt` est recopié, mais pas le tampon pointé par `pt`. Ensuite, le programme détruit l'objet `str`. Le tampon est aussi détruit.



L'objet copié pour le retour est erroné car le pointeur `pt` pointe sur une zone libérée ! Le constructeur de copie aurait dû donner :



Le plus grave est que cela passe généralement inaperçu car les données du tampon ne sont pas physiquement détruites. Si une version de debug de `::delete` remplit les zones à effacer avec une valeur quelconque avant de libérer la mémoire, cette erreur apparaîtra lors de l'utilisation du tampon. Vous pouvez vous en convaincre en ajoutant avant le `"delete [] pt"`, la commande `"*pt='\0';"`, afin de le détruire. Cette erreur se présente aussi dans un autre cas.

```
void main()
{
    CString a="abc";
    {
        CString b="def";
        a=b; // Copie des pointeurs !
    }
    // Erreur ici
}
```

Cette fois, c'est l'opérateur d'affectation par défaut qui la gère. Lors de la destruction de l'objet `b`, `a` devient erroné. Dans le cas présent, il faut ajouter :

```
private:
    CString(const CString& x);    // Non implémenté
    CString& operator =(const CString& x); // Non implémenté
```

pour que le compilateur signale le problème. Une solution élégante pour régler ce cas de figure est d'utiliser un pointeur d'agrégation comme attribut de la classe.

```
class CString
{
    CPtrArrayAggr<char> pt;
public:
    CString(const char* src)
    : pt(new char[strlen(src)+1])
    {
        strcpy(pt,src);
    }
};
```

Avec cet outil, l'affectation, le constructeur de copie et le destructeur sont correctement générés par le compilateur. Toujours déclarer, pour tout objet ayant un pointeur, le constructeur de copie et l'opérateur d'affectation. La troisième question traite des différentes allocations mémoire.

Question 3

```
class CString
{
    char* pt;
public:
    CString()
    {
        pt=(char*)::malloc(4);
        ::strcpy(pt,"ABC");
        cout << "CString::CString()" << endl;
    }
    ~CString()
    {
        ::free(pt);
        cout << "CString::~CString()" << endl;
    }
};

void main()
{
    CString* pt=new CString[3];
    // ...
    delete pt;
}
```

CString

Qu'affiche *main*, pourquoi et comment corriger ?

main affiche :

```
CString::CString()
CString::CString()
```

```
CString::CString()
CString::~CString()
```

Il n'y a plus de parité entre les constructeurs et les destructeurs !

Lors de la création dynamique d'un tableau d'objets, le compilateur alloue la mémoire nécessaire, puis appelle le constructeur vide pour chaque élément du tableau.

Lors de la destruction de celui-ci, il ne fait pas la différence entre un pointeur sur un objet et un pointeur sur un tableau d'objets. Il faut lui indiquer dans la syntaxe de `delete` qu'il s'agit d'un tableau. Dans ce cas, le destructeur est appelé pour chaque élément du tableau avant que la mémoire ne soit libérée.

Il faut indiquer dans le programme l'instruction `delete [] pt`. Dans cet exemple, le programme laisse, dans le meilleur des cas, de la mémoire perdue ou, dans le pire, cassera le chaînage du tas. L'erreur n'apparaîtra que très tardivement, lorsqu'il n'y aura plus de mémoire ! Une version de debug de `new` et de `delete` peut la signaler en vérifiant la validité d'un pointeur avant d'effectuer un `delete`. Si vous appelez `delete pt` à la place de `delete [] pt`, le pointeur effacé n'est pas sur une adresse retournée par un `new` précédent. Ceci est le résultat du compteur caché, ajouté par le compilateur. Cette particularité peut être détectée à l'exécution.

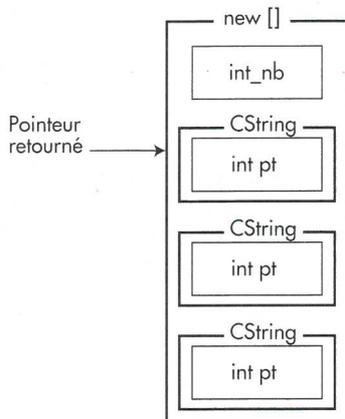
Dans les anciennes versions de C++, il fallait indiquer dans les crochets du `delete` le nombre d'éléments à effacer. Maintenant, depuis 1992, lors de la création du tableau, le compilateur mémorise le nombre d'éléments de celui-ci. Lors de l'effacement, le compilateur regarde ce nombre afin d'appeler correctement tous les destructeurs. Il n'est plus nécessaire d'indiquer le nombre d'éléments lors du `delete`, le compilateur n'en tient d'ailleurs plus compte.

La création et la destruction d'un tableau sont traduits à peu près comme ceci :

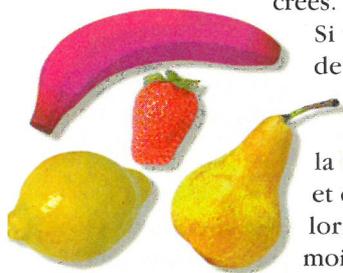
```
void main()
{
    // CString* pt=new CString[3]; Allocation du tableau
    CString* pt=
        (CString*)::new(sizeof(int)+ // Ajoute un entier au début
        sizeof(CString)*3);
    if (pt==NULL)
    {
        *(int*)pt=3; // Nb d'éléments du tableau
        pt=(CString*)((int*)pt+1); // Avance le pointeur
        for (int i=0;i<3;++i) // Appelle le constructeur
            pt[i].CString(); // pour tous les éléments
    }
    // delete [] pt; Libération du tableau
    if (pt==NULL)
    {
        int j=((int*)pt-1); // Nb d'éléments du tableau
        for (int i=j;i>=0;-i) // Appelle le destructeur
            pt[i].~CString(); // pour tous les éléments
        pt=(CString*)((int*)pt-1); // Recule le pointeur
        ::delete(pt); // Efface la zone pointée
    }
}
```

Ce qui se présente en mémoire comme suit.

Il est de la responsabilité du programmeur d'appeler la bonne version de `delete` suivant l'objet manipulé, sinon, seul le premier destructeur est appelé. Il est à noter que si une exception est générée à la construction d'un des éléments du tableau, le mécanisme de remontée de la pile appelle les destructeurs uniquement pour les objets déjà créés.



Si vous convertissez un pointeur lors de l'allocation, il faut faire de même lors de la destruction. Par exemple, il est courant d'allouer la taille nécessaire à plusieurs objets, et d'appeler le constructeur de celui-ci lors de l'utilisation de cette zone mémoire.



```
{
  CObj *tab=(CObj*)new char[sizeof(CObj)*10]; // Reserve la place
  int i=0;

  // ...
  tab[i].CObj::CObj(); // Construction d'un des éléments du tableau

  delete [] tab; // Erreur
}
```

Dans ce cas, l'exécution de `delete [] tab` est erronée, car le compilateur va appeler un nombre aléatoire de destructeurs des objets soi-disant présents dans le tampon. Ils n'ont pas tous été construits. Il faut corriger le programme comme cela.

```
for (int j=max-1; j>=0; --j)
{
  tab[j].CObj::~CObj(); // Appel des destructeurs
}
delete [] (char*)tab; // Effacement de la réserve
```

Toujours utiliser `delete []` pour les objets alloués avec un `new []`.

Le mois prochain nous parlerons des trois questions dont vous avez, lecteur fidèle, la primeur.

Question 4

```
void main()
{
  char* pt;

  pt=new char [10];
  // ...
  delete pt;
}
```

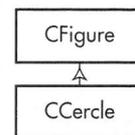
C'est incorrect, pourquoi ? (*char* n'est pas un objet!)

Question 5

```
class CFigure
{
public:
  virtual void trace()
  {
    cout << "CFigure::trace()" << endl;
  }
  ~CFigure()
  {
    cout << "CFigure::~CFigure()" << endl;
  }
};

class CCercle : public CFigure
{
public:
  virtual void trace()
  {
    cout << "CCercle::trace()" << endl;
  }
  ~CCercle()
  {
    cout << "CCercle::~CCercle()" << endl;
  }
};

void main()
{
  CFigure* p;
  p=new CFigure();
  p->trace();
  delete p;
  p=new CCercle();
  p->trace();
  delete p;
}
```



Qu'affiche *main*, pourquoi, et comment corriger ?

Question 6

```
class CHopital
{
public:
  void afficheNom()
  {
    cout << "CHopital::afficheNom()" << endl;
  }
};

void main()
{
  void (*ptf)(void); // Pointeur de fonction

  ptf=&CHopital::afficheNom;
  ptf(); // Appel de CHopital::f()
}
```



Ce n'est pas compilable, pourquoi, et comment corriger ?

Vous avez un mois pour trouver la réponse à chacun de ces problèmes, bonne recherche et bonne migraine! ;o

Ne jetez plus l'argent par les fenêtres

Les applications modernes se doivent d'avoir des interfaces utilisateurs sophistiquées. Malheureusement celle-ci sont trop souvent coûteuses à écrire.

Frédéric Mazué
frederic.mazue@wanadoo.fr



l'ation sont toujours trop importants et l'utilisation de tels outils propriétaires met à mal la portabilité de C++. En supposant que l'on écrive une application devant tourner sur trois plates-formes on peut craindre que le temps pris pour la création et le portage de l'interface prenne de cinq à dix fois le temps pris pour écrire le cœur de l'application.

La solution Python : rapidité de développement et portabilité

À une époque encore peu lointaine, l'informatique était une affaire d'initiés ou de passionnés qui s'accommodaient des interfaces utilisateurs les plus frustes. De nos jours, l'utilisateur de l'ordinateur n'est plus forcément un informaticien pur et dur, il réclame des programmes aux interfaces graphiques (GUI) simples, agréables et intuitives.

Un travail pénible et coûteux

Le développement d'interfaces utilisateurs est souvent la plaie des développeurs. Ceux-ci préfèrent en général se concentrer sur la résolution des problèmes, sur l'élaboration d'algorithmes performants. Leur écriture est également très coûteuse, surtout dans le cas d'applications écrites en C++. Le choix de ce langage pour écrire une application s'avère bien souvent judicieux. L'orientation objet du C++ permet une programmation élégante et les performances à l'exécution sont assurées. L'affaire se gâte en ce qui concerne la création des interfaces. La perte de temps engendrée par les multiples et trop longues compilations a de quoi épuiser les patiences les plus angéliques et les budgets les plus confortables. Certes, il existe des environnements de développement visuel comme C++Builder sous Windows qui facilitent énormément la tâche. Malgré tout, les temps de (re)compilation

Python est un langage de conception très moderne. Avant tout, il a été conçu de façon à ce que le temps d'écriture des applications soit réduit au minimum. Ensuite, il est portable sur toute plate-forme disposant d'un compilateur C. Un script Python s'exécute n'importe où sans le moindre problème, but que Java n'a pas encore vraiment atteint. Existe-t-il une JDK pour Amiga par exemple? Python est un langage compilé de manière transparente à l'utilisateur, ce qui lui assure une rapidité d'exécution très honorable. Du point de vue du programmeur, Python est interprété : on écrit le code et on l'essaie immédiatement. Mais le plus grand coup de génie de Guido van Rossum, le père de Python, est sans aucun doute de s'être abstenu de vouloir créer un énième langage à tout faire. Au lieu de cela, Python a été conçu comme facilement extensible. Rien n'est plus simple que d'écrire, en C ou en C++, un module d'extension à Python. Mieux que cela, il existe maintenant des outils qui font le travail automatiquement. Il est même possible d'envelopper une bibliothèque C++ entière dans un module Python en levant à peine le petit doigt. Voir par exemple à <http://www.swig.org>. Bref, Python reste un langage léger, idéal pour faire la « colle » entre des morceaux de codes C/C++. De nombreux développeurs ont exploité ces facilités pour écrire des modules d'extension « GUI » portables. Grâce à tout ceci, la stratégie pour créer des applications sans perte de temps et d'argent se dessine facilement : on écrit le cœur de l'application en C++ pour obtenir les

meilleures performances. L'écriture du code sera rapide puisque le développeur est déchargé des soucis d'interface et le temps de compilation sera réduit au minimum. Le code sera portable moyennant une simple recompilation des sources. Ensuite, on fait de ce code un module d'extension à Python, puis on l'habille d'un GUI écrit grâce à un autre module d'extension Python. Prenant le meilleur des deux langages, on gagne sur tous les tableaux. Temps de développement réduit, performance à l'exécution et surtout, cerise sur le gâteau, l'application hybride ainsi créée se retrouve immédiatement portable sans même avoir besoin de s'en préoccuper. Il existe de nombreuses extensions GUI à Python. Nous allons en présenter deux: Tkinter et wxPython.

Tkinter

Tkinter est historiquement le premier GUI Python. Écrit par Steen Lumholt et Guido van Rossum, Tkinter est en fait une encapsulation de Tk, lui-même créé par John Ousterhout. Tk est un ensemble de scripts écrits en langage Tcl. Tkinter fait normalement partie du paquetage Python. Tout script Python faisant appel au module d'extension Tkinter tournera sans qu'il soit besoin de changer une ligne de code, pourvu que la plate-forme cible dispose de Tcl/Tk.

Tkinter est un GUI qui est un plus petit dénominateur commun des contrôles graphiques que l'on trouve sur différentes plates-formes. Guido van Rossum a créé Tkinter dans la philosophie de Python ce qui veut dire que la programmation est en très aisée et rapide. La mise en place des fonctions de rappels (*callback*) pour le traitement des événements est un jeu d'enfant et l'orientation objet donnée par Python à Tk permet la réutilisation du code. Les figures 1 et 2 montrent une petite application tournant sous Linux et Windows sans qu'il soit nécessaire de changer la moindre ligne de code.



Figure 1: notre exemple Tkinter sous Linux.

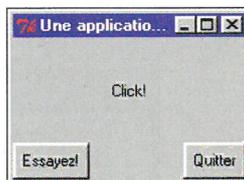


Figure 2: notre exemple Tkinter sous Windows.

Voici le code de cette application :

```
#!/usr/bin/env python

from Tkinter import *
root = Tk()
root.title('Une application Tkinter')
label = Label(root, text = 'Bonjour')
label.pack(side=TOP, padx=70, pady=30)
bouton1 = Button(root, text='Essayez!', command=lambda x=label: x.config(text='Click!')).pack(side=LEFT)
bouton2 = Button(root, text='Quitter', command = root.quit).pack(side=RIGHT)
root.mainloop()
```

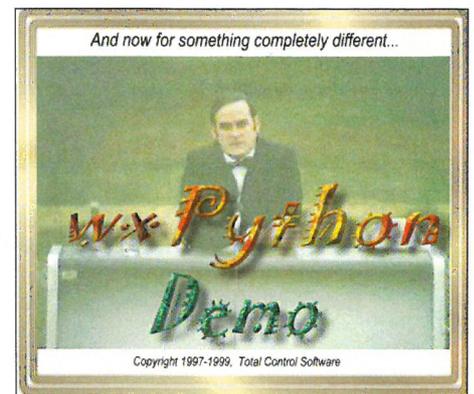
Ainsi, nous voyons qu'une petite dizaine de lignes de code sont suffisantes pour ouvrir une fenêtre, y placer un label et deux boutons, et gérer les actions utilisateurs sur ces bou-

tons. Une action sur le premier bouton modifie le contenu du label tandis qu'une action sur l'autre bouton ferme l'application. Ce programme a été écrit en deux minutes et est directement portable sur Linux, Windows, Macintosh, Solaris et plus généralement toute plate-forme UNIX, etc. Qui dit mieux ?

Bien sûr, on peut voir quelques désavantages. Ce type d'application est plus difficile à déployer qu'un simple exécutable C++ puisque Tcl/Tk doivent être présents sur la machine cible. D'autre part Tkinter n'est pas un GUI très riche et le traitement des événements n'est pas aussi complet en comparaison de ce que proposent d'autres GUI. On peut également reprocher à Tkinter de ne pas être très bien documenté. Mais Tkinter fait merveille lorsqu'il s'agit de prototyper des applications. Son maniement facile, son gestionnaire de mise en forme astucieux et la concision du code permettent d'écrire en une après-midi ce qui pourrait demander une semaine de travail avec d'autres moyens.

wxpython

wxPython est quant à lui, un module d'extension construit autour de la bibliothèque C++ wxWindows dont il apporte toute la puissance tout en affranchissant des fastidieuses opérations de compilation. Avant de parler de wxPython proprement dit, décrivons brièvement wx-



Essayez wxPython et appréciez la différence!

Windows. Le projet wxWindows (<http://www.wxwindows.org>) a vu le jour en 1992 à l'initiative de Julian Smart qui déplorait que le monde du C++ souffre cruellement du manque d'une bibliothèque multi-plates-formes permettant l'écriture de GUI de qualité. Le projet wxWindows, démarré dans l'enthousiasme, a stagné un moment pour reprendre de plus belle. Aujourd'hui, ce cadre C++ est d'une richesse

absolument incroyable. Contrairement à d'autres cadres C++ de ce genre, wxWindows ne présente pas un plus petit dénominateur commun des contrôles que l'on peut trouver sur diverses plates-formes. Au contraire, tous les contrôles sont encapsulés, et pour faire une comparaison, wxWindows est d'une richesse supérieure aux deux cadres C++ leaders sur la plate-forme Microsoft Windows, à savoir MFC de Microsoft et VCL de Bor-

land. Par exemple, la classe encapsulant les images gère les formats JPEG, PNG, TIFF, PCX, GIF, PNM et BMP, ridiculisant ainsi les classes proposées par MFC et VCL. Quand un

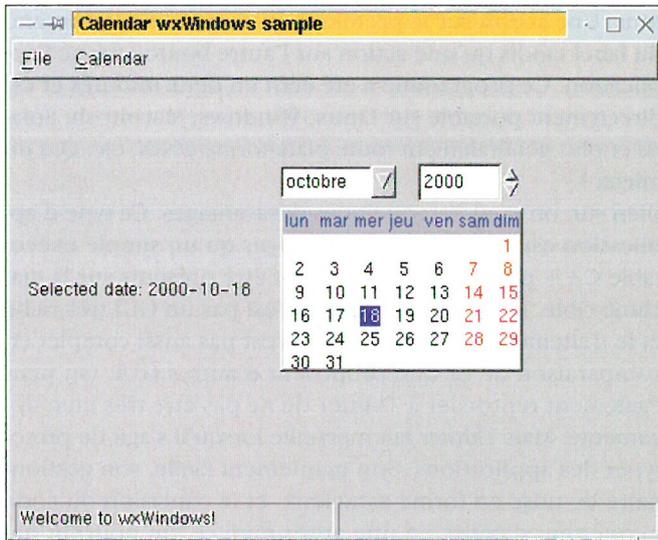


Figure 3: wxWindows propose des contrôles originaux comme ce calendrier.

```

#!/usr/bin/env python

from wxPython.wx import *

# ----- class MyApp -----
class MyApp(wxApp):
    def OnInit(self):
        frame = MyFrame(None, -1, "Une application wxPython")
        frame.Show(True)
        self.SetTopWindow(frame)
        return True

# ----- classe MyFrame -----
class MyFrame(wxFrame):
    def __init__(self, parent, id, title):
        wxFrame.__init__(self, parent, id, title)
        self.SetDimensions(width=200, x=20, y=20, height=150)
        self.Center()
        self.panel = wxPanel(self, -1)
        self.label = wxStaticText(self.panel, -1, 'Bonjour', wxPoint(70, 50))
        self.id1 = wxNewId()
        self.id2 = wxNewId()
        self.bouton1 = wxButton(self.panel, self.id1, \
            'Essayez!', wxDefaultPosition)
        self.bouton2 = wxButton(self.panel, self.id2, \
            'Quitter', wxDefaultPosition)
        self.bouton1.SetPosition(wxPoint(10, self.GetClientSize().GetHeight() \
            - self.bouton1.GetSize().GetHeight()-5))
        self.bouton2.SetPosition(wxPoint(110, self.GetClientSize().GetHeight() \
            - self.bouton2.GetSize().GetHeight()-5))
        EVT_BUTTON(self, self.id1, self.Bouton1Handler)
        EVT_BUTTON(self, self.id2, self.Bouton2Handler)

    def Bouton1Handler(self, event):
        self.label.SetLabel('Click !')

    def Bouton2Handler(self, event):
        self.Destroy()

# -----
app = MyApp(0)
app.MainLoop()

```

contrôle n'existe pas sur une plate-forme, il est émulé. Des contrôles originaux comme le calendrier montré **figure 3** augmentent encore la richesse de la bibliothèque. Les spécificités de chaque plate-forme ne sont pas négligées pour autant. Ainsi wxWindows dispose de classes encapsulant COM et ODBC sous Microsoft Windows. wxWindows existe pour Linux, Windows 9x et Windows NT et toute plate-forme UNIX équipée de Gtk+, Motif, Lesstif et Macintosh. Une application wxWindows est très facilement portable. Il suffit seulement de recompiler les sources pour la plate-forme cible sans devoir changer une ligne de code.

Encore plus loin

wxPython se présente comme une enveloppe (un *wrapper*) de wxWindows dans un module d'extension à Python. wxPython est le résultat du travail de Robin Dunn (voir à <http://www.wxpython.org>). Construit au moyen de Swig dont nous avons parlé plus haut, wxPython est principalement constitué de code C++, ce qui assure une rapidité d'exécution sans reproche. wxPython prend en charge toutes les fonctionnalités de wxWindows, y compris les fonctionnalités spécifiques à des plates-formes telles COM ou ODBC, mais apporte encore quelques améliorations très pertinentes. Prenons un exemple : wxWindows dispose de gestionnaires de mise en forme à la manière de ceux que l'on trouve avec Java. La programmation de ces gestionnaires est assez difficile. Fidèle à l'esprit de Python qui se veut un langage de développement rapide, Robin Dunn a ajouté des gestionnaires de mises en forme construits sur ceux de Wx-Widgets mais plus rapides à programmer. Pour vous convaincre de la puissance de wxPython, essayez le programme de démonstration fourni avec le module. La **figure 4** est une capture d'écran du gestionnaire de mise en forme incorporé à wxPython et mis en œuvre par ce programme de démonstration.

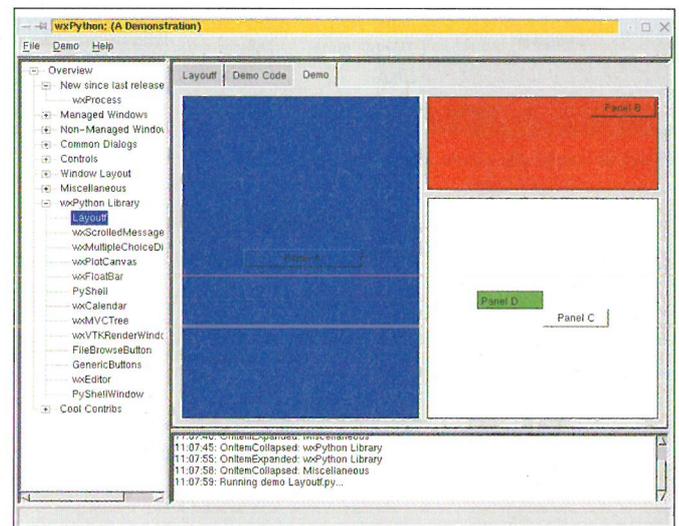


Figure 4: les gestionnaires de mise en forme de wxPython.

Le listing ci-contre ouvre la même fenêtre que l'exemple Tkinter, voir les **figures 5 et 6**.

On notera avec quelle facilité les gestionnaires d'événements des boutons sont installés dynamiquement grâce à la fonction `EVT_BUTTON`. Reportez-vous au code sur le [Listing 1](#).

Quelques remarques : même si nous constatons qu'il a fallu écrire un peu plus de code qu'avec Tkinter pour parvenir au même résultat, ce code reste très clair et concis. Il ne m'a fallu que cinq minutes pour l'écrire. Par contre, le déploiement des applications est simplifié : wxPython est une simple DLL sous Windows ou un simple paquetage RPM sous Linux. Par ailleurs, le code wxPython présente une ressemblance frappante avec le code wxWindows/C++, et nous pouvons en tirer parti. Ainsi, même si l'on ne souhaite pas écrire une application hybride partie Python – partie C++, c'est une bonne idée d'écrire un petit script permettant de prototyper le GUI, d'essayer ce script sur diverses plates-formes pour se rendre compte de visu du résultat, puis, profitant de la similitude entre wxPython et wxWindows réécrire rapidement le code en C++. Faites l'expérience et vous constaterez qu'au final, l'interface aura été créée beaucoup plus rapidement qu'elle ne l'aurait été avec le seul C++. Autre avantage déci-

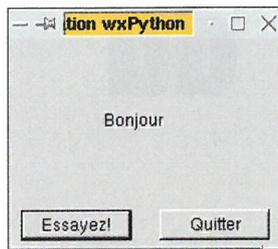


Figure 5: notre exemple wxPython sous Linux.

sif : wxWindows/wxPython sont bien documentées. wxPython et wxWindows sont si proches que leur documentation est commune, à très peu de choses près. Il n'est donc besoin de bien connaître qu'un seul cadre et une seule documentation pour développer rapidement pour Linux, UNIX, Windows. Tout programmeur qui a passé un « certain temps » à étudier des tonnes de documentations rébarbatives pour savoir comment repeindre une fenêtre sur telle ou telle plateforme avec C++ comprendra facilement ce que je veux dire.

Pour conclure

La création d'interfaces graphiques est un travail coûteux et souvent fastidieux. Lorsqu'il s'agit de réduire les coûts de développement sans sacrifier à la qualité, l'écriture d'applications hybrides Python/C++ est une solution de 1^{er} choix qui prend le meilleur de deux langages formidables. Même si l'on préfère des applications 100% C++, le prototypage des interfaces graphiques avec Python fait gagner énormément de temps et donc d'argent. Sans compter la portabilité qui vient automatiquement. ■

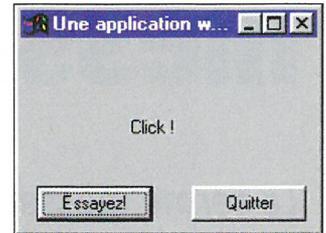


Figure 6: notre exemple wxPython sous Windows.

Spécial Gravure !

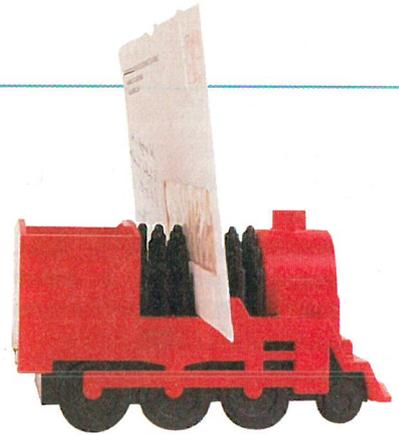
De la théorie à la pratique, du CD Audio au DVD, les conseils et les astuces pratiques pour bien réussir sa copie.

Offerts :

1 CD-R vierge Carbon CD de Hi-Space

ACTUELLEMENT EN VENTE CHEZ VOTRE MARCHAND DE JOURNAUX

Les concepts du messaging



Le terme *messaging* désigne différents procédés de programmation distribuée basés sur l'échange de messages. Le *messaging* est à la programmation distribuée A2A (Application to Application) ce que le mail est à la communication H2H (Human to Human), les principes sont les mêmes seuls les outils changent.

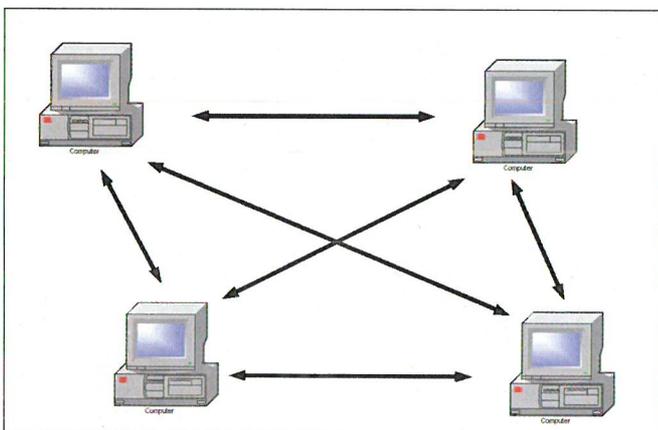
Médéric Morel, Neoxia

Le *messaging* est un procédé assez ancien et solidement éprouvé qui a notamment fait ses preuves dans la communication entre architectures hétérogènes, particulièrement entre les *mainframes* et les micro-ordinateurs.

Les messages

Tout d'abord, délimitons la notion de message. Il s'agit d'un bloc de données quelconques (texte, image, son, ...) de taille connue (par opposition à un flux) qui sert de support pour la communication entre plusieurs applications. La taille des messages est souvent limitée par les outils utilisés et la capacité du réseau, la limite est généralement fixée à plusieurs Méga octets (Mo). On peut donc comparer la notion de message à celle de courrier (*mail*).

Un message comporte généralement deux ou trois parties distinctes. La première partie est composée du *header* (entête), qui contient les informations nécessaires à l'acheminement du message telles que l'expéditeur, le destinataire et des *flags* qui précisent les options de transport comme les accusés de réception, par exemple.



Le *messaging peer-to-peer*.

Dans certains produits de *messaging*, on trouve aussi une partie dédiée au stockage de métadonnées liées au message (sortes de propriétés attachées au message) qui permettent d'opérer un filtrage sur un ensemble de messages. Enfin, on trouve le contenu proprement dit du message.

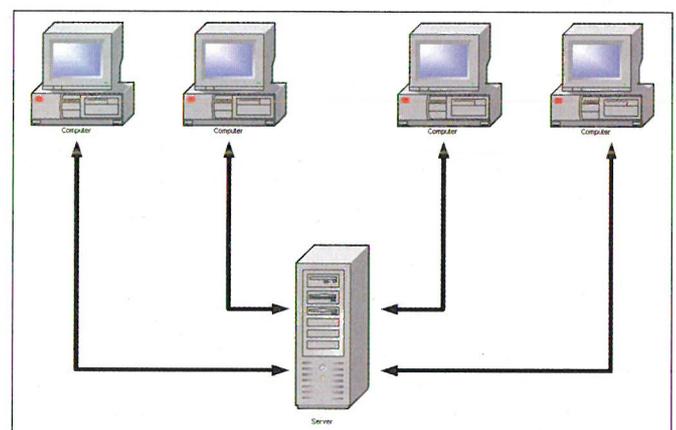
Les modes de messaging

Il existe deux manières différentes de faire du *messaging* :

- Le *messaging peer-to-peer*

Il s'agit d'un mode de communication où les applications communiquent directement entre elles sans passer par un tiers chargé du transport des messages.

Ce mode a l'inconvénient de nécessiter que les applications se connaissent et soient en permanence à l'écoute de l'arrivée de nouveaux messages. Cette condition s'avère particulièrement problématique lorsque le nombre d'applications est important. Cet article ne couvre pas le *messaging* en mode *peer-to-peer* en raison du fait que celui-ci est très peu utilisé.



Le *messaging avec un middleware*.

• Le *messaging* basé sur un *middleware*

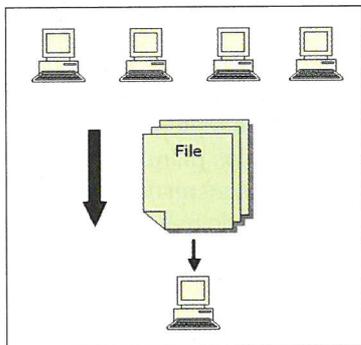
Dans ce mode, les applications ne se transmettent pas directement leurs messages, mais passent par un tiers appelé MOM (*Message Oriented Middleware*) ou encore *broker* de messages, qui se charge du transport. On peut comparer le MOM aux services de la poste ou d'un transporteur. L'utilisation d'un tel *middleware* apporte une grande souplesse d'utilisation et permet notamment la communication asynchrone.

La diffusion des messages

Une fois choisi un MOM, il existe différentes façons d'échanger des messages entre applications. On retient principalement trois procédés de communication :

• Le mode point à point.

En mode point à point, les messages sont échangés par l'intermédiaire d'une entité appelée *Queue* ou file d'attente. Les messages sont déposés par leur expéditeur dans la file d'attente où ils sont stockés en attendant d'être retirés par le destinataire. Un nombre quelconque d'applications peuvent utiliser une file d'attente que cela soit pour y déposer des messages ou pour en retirer. C'est la version A2A (*Application to Application*) des protocoles de messagerie électroniques telles que SMTP, POP3 et IMAP4.



Le mode point-à-point.

En résumé, le mode point à point possède les caractéristiques suivantes :

FIFO (*First In First Out*), car les messages apparaissent dans la file d'attente dans leur ordre d'arrivée.

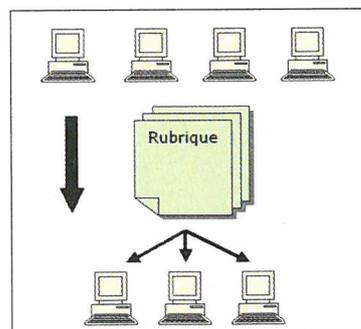
Unicast, car chaque message n'est traité que par un seul destinataire.

Asynchrone, puisque l'expéditeur dépose le message dans la file sans savoir par qui et quand le message sera traité.

• Le mode publication/abonnement.

En mode publication/abonnement (*publish/subscribe*) les messages sont échangés par l'intermédiaire d'une entité appelée *topic* ou rubrique.

Les messages sont déposés par l'expéditeur dans la rubrique. Contrairement à ce qui se passe dans le cas du mode point à point où les messages ne sont délivrés qu'à un seul destinataire, ici les messages sont expédiés à tous les abonnés de la rubrique. C'est la



Le mode publication/abonnement.

version A2A (*Application to Application*) des protocoles de newsgroup comme NNTP.

En résumé le mode publication/abonnement possède les caractéristiques suivantes :

FIFO (*First In First Out*), car les messages apparaissent dans la rubrique dans leur ordre d'arrivée.

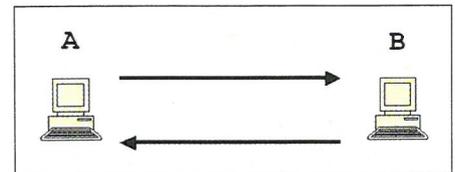
Multicast, car chaque message est délivré à chacun des abonnés de la rubrique.

Asynchrone, puisque l'expéditeur dépose le message dans la rubrique sans savoir par qui et quand le message sera reçu.

• Le mode requête/réponse.

Ce mode est très proche du mode point à point car la transmission des messages se fait par l'intermédiaire d'une file d'attente. La différence entre les deux modes réside dans le fait qu'en mode requête/réponse l'expéditeur (A) attend une réponse à son message. La réponse prend, bien sûr, la forme d'un message qui lui est adressé par l'application qui a traité son message (B).

Généralement la réponse est envoyée sur une file d'attente temporaire créée pour l'occasion et que seul l'expéditeur écoute.



Le mode requête/réponse.

En résumé le mode requête/réponse possède les caractéristiques suivantes :

FIFO (*First In First Out*) car les messages sont distribués dans leur ordre d'envoi.

Unicast, car chaque message n'est traité que par un seul destinataire.

Synchrone, puisque l'expéditeur attend une réponse à son message avant de continuer ses traitements.

Cas d'utilisation

Chacun des trois modes de diffusion correspond à des cas précis d'utilisation.

Le mode point à point est surtout utilisé pour déclencher des traitements à distance sans qu'une réponse soit attendue, c'est par exemple le cas du module de saisie d'une application de comptabilité, l'opérateur de saisie remplit le formulaire et l'application poste le formulaire sur une file d'attente où il sera ensuite traité.

Le mode publication/abonnement est adapté à la propagation d'événements, il peut notamment être utilisé pour envoyer des alarmes, gérer des *logs*, ou diffuser des mises à jours de données.

L'intérêt de ces deux modes réside dans le faible couplage entre les applications ce qui permet de faire communiquer facilement des applications très différentes. De plus le caractère asynchrone des appels permet des traitements en différé des messages.

Références

IBM MQ Series

<http://www-4.ibm.com/software/ts/mqseries/>

Microsoft MSMQ

http://msdn.microsoft.com/library/psdk/msmq/msmq_overview_4ilh.htm

Tibco Rendezvous

<http://www.tibco.com/products/rv/index.html>

Progress Software Sonic MQ

<http://www.progress.com/sonicmq/index.htm>

BEA MessageQ

<http://www.bea.com/products/messageq/datasheet.shtml>

Borland VisiMessage

<http://www.borland.com/devsupport/appserver/downloads/visimessage/visimessengerel.html>

Un pont entre MSMQ et MQSeries

http://msdn.microsoft.com/library/psdk/his/bridge1_5vx0.htm

Le mode requête/réponse est d'un intérêt plus limité car même s'il semble rétablir la continuité des traitements, l'appel étant traité de manière synchrone, c'est au prix de la perte de la possibilité de traitements en différé. De plus, la continuité du traitement est toute relative puisqu'il n'est pas possible de propager des contextes transactionnels par ce moyen (voir le paragraphe sur le *messaging* et les transactions).

Messaging et répartition de charge

Une autre application intéressante du *messaging* est la répartition de charge (*load balancing*). En effet, le mode point à point présente naturellement cette propriété. Voici un exemple : prenons une application où des factures sont saisies puis envoyées pour traitement sur une file de messages, une autre application se charge de prendre les messages dans la file et de faire les traitements appropriés. Si, pour une raison quelconque, le nombre d'opérateurs de saisie devait augmenter et donc, le nombre de messages postés dans la file, on risquerait alors de voir l'application de traitement des messages saturée. Pour résoudre ce problème il suffit alors simplement de rajouter une machine hébergeant l'application de traitement. Puisque les messages sont extraits d'une file, chaque message ne peut être lu qu'une seule fois. Donc, la seconde machine, en prenant des messages dans la file, soulage la charge de la première. On peut bien-sûr installer plus de deux machines.

Ce procédé est le moyen le plus simple de faire du *load-balancing*, il présente par ailleurs des aptitudes au *fail-over* (tolérance de pannes) car en cas de défaillance d'une machine de traitement, les autres prennent le relais automatiquement en traitant plus de messages.

Messaging et transactions

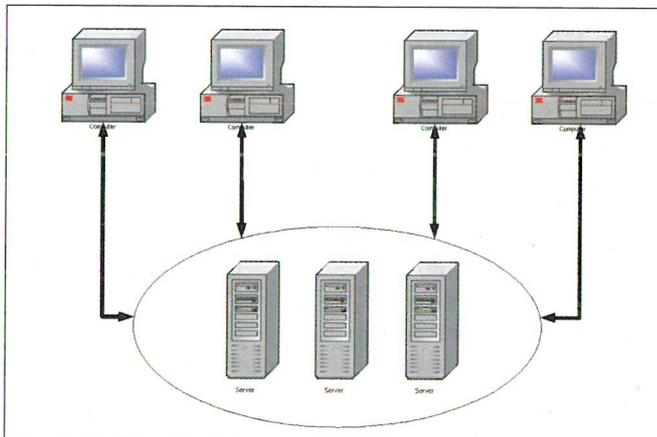
La notion de transaction est généralement attachée aux bases de données mais elle existe aussi dans le cas des MOM. Les transactions sur messages s'appliquent à la fois en lecture et en envoi. Pendant la durée d'une transaction, toutes les opérations de lecture de message et d'envoi de messages sont simulées par le MOM, c'est seulement lors du *commit* que les messages sont véritablement envoyés et/ou définitivement reçus c'est-à-dire enlevés de la file de message, en cas de *rollback* toutes les opérations sont bien-sûr annulées. La transaction des messages en lecture peut poser des problèmes si le traitement d'un message occasionne systématiquement des plantages dans l'application réceptrice, certaines implémentations de MOM offrent des services évolués permettant de traiter ces cas.

Il est important de retenir que même si les MOM offrent un support des transactions sur message, ils ne permettent en aucun cas, de propager un contexte transactionnel par l'intermédiaire d'un message.

Les dernières versions des produits commerciaux de type MOM voient l'apparition du support du protocole XA pour les transactions. Ce support permet de faire participer un MOM à une transaction distribuée qui implique d'autres ressources (SGBDR, EJB, ...)

Messaging et clustering

Enfin, certains produits permettent de monter les MOM en *cluster*, il s'agit d'un procédé qui consiste à lancer plusieurs instances du *broker* de messages sur le réseau pour éviter une paralysie des applications si le *broker* venait à être indisponible. Bien évidemment, ces instances multiples du *broker* de messages doivent partager les mêmes entités (files, rubriques) et utiliser la même source de stockage (*backing-store*) pour les messages (généralement un SGBDR).



Un cluster de MOM.

Les produits du marché

Il existe un nombre important de *brokers* de messages sur le marché, certains sont disponibles sur plusieurs plates-formes, d'autres non. Il en va de même pour le support du *clustering*. Certains produits se détachent néanmoins nettement des autres, en particulier MQ Series d'IBM qui fonctionne sur un très grand nombre de plates-formes (AIX, Solaris, Windows, *Mainframes*, ...). Du côté de Microsoft, MSMQ (*Microsoft Message Queue Server*) est distribué en standard sous Windows 2000 et en option pour les autres versions de Windows. On peut également citer les produits de Tibco, Rendez-Vous (Tibco RV), très utilisé dans la finance et ceux de Progress Software, SonicMQ, une implémentation d'un *broker* de message pour Java. Enfin, certains éditeurs de serveurs d'applications fournissent en standard ou en option des *brokers* de messages, on peut citer notamment BEA MessageQ et Borland VisiMessage.

Il existe aussi des ponts permettant de rendre certains de ces MOM interopérables.

Simplicité complexe

Les brokers de messages sont des outils relativement simples à mettre en œuvre et qui permettent la communication entre applications et plates-formes hétérogènes. Fiables et ro-

bustes, ils peuvent être utilisés comme glue entre plusieurs applications utilisant des technologies différentes. De plus, ils peuvent communiquer et participer à des transactions avec d'autres outils utilisés dans l'entreprise comme les bases de données et les serveurs d'application.

Mais cette simplicité apparente ne doit pas dissimuler des difficultés spécifiques liées à l'architecture sous peine de graves problèmes de performances et d'évolutivité. Le choix de messages à petit ou gros grain, l'utilisation ou non des transactions, les besoins éventuels d'interopérabilité avec des serveurs d'applications (EJB, MTS), le choix du *backing-store* (lieu de stockage des messages) ou encore du protocole de transport utilisé (IP, IIOP, DCOM, ...) sont autant d'étapes cruciales pour la pérennité de la solution.

Dans les prochains articles sur le sujet nous aborderons en détail JMS, la solution retenue par Sun pour la collaboration entre les applications Java et les *brokers* de messages, et la vision du *messaging* par Microsoft à travers la plate-forme .NET et le produit MSMQ.

L'auteur :

Médéric MOREL est consultant chez Neoxia un cabinet de conseil en architectures distribuées.

www.neoxia.com

Exploitez le meilleur de Linux

Lotus Domino Server
XFree86 4.02, Sun Open Office, IBM Java Tools...

Traitement de l'image :
de l'acquisition à la manipulation en PHP

Linux+

Exploiter le meilleur de Linux

En partenariat avec LINUX JOURNAL (USA)

Janvier 2001 - 36 F

PME en réseau : les solutions économiques

PRATIQUE
Installer Linux sur un portable
Automatiser les sauvegardes
Gérer les paquets Debian

MULTIMÉDIA
Plusieurs écrans sur votre serveur
Musique à la demande

CAHIER EXPERT
Détection et prévention des intrusions
Configurer un système RAID
Clustering avec PVFS

"Outre l'actualité du logiciel libre et de l'Open Source, Linux+ vous offre, chaque mois, une information pratique pour découvrir ou perfectionner vos connaissances sur Linux et ses principales applications : bureautique, graphisme, Internet, réseau,...

Un cahier Expert vous propose par ailleurs des articles plus techniques sur les aspects serveurs, bases de données et programmation...

Chaque mois, sur le CD-Rom, des programmes complets, utilitaires, sharewares, jeux...

En vente actuellement chez votre marchand de journaux

Une excursion

en mémoire

Dépasser le nombre de processus maximum des noyaux Linux i386.

Par Zhang Yong - Linux Journal
Traduction Frédéric Mazué

La gestion des processus est la partie la plus importante des systèmes d'exploitation. La qualité de sa conception et de son implémentation affecte grandement les performances. Dans un système d'exploitation multi-processus, plusieurs programmes s'exécutent simultanément ce qui augmente l'emploi du CPU et la charge système. Avec l'exécution simultanée de processus, nous offrons de multiples services et nous pouvons servir plus de clients en même temps, ce qui est la tâche principale d'un système d'exploitation moderne.

Dans l'architecture Linux Intel i386, le multi-processus est déjà supporté. Grâce à un choix d'algorithmes performants de planification des tâches, le noyau a un temps moyen de réponse assez bas et des performances système relativement hautes. Mais, malheureusement, il y a une limitation dans le noyau 2.2.x qui fait que le nombre maximum de processus simultanés est de 4090. Ce nombre est suffisant pour un ordinateur de bureau, mais n'est pas adéquat pour un serveur d'entreprise.

Considérons le principe de base d'un serveur web typique basé sur une technologie multi-processus/*multi-threads*. Quand une requête client arrive, le serveur web crée un processus enfant ou un *thread* pour traiter la requête. Ainsi, il est facile pour un serveur fortement chargé d'avoir des milliers de processus en cours d'exécution. En fait, la plupart de tels serveurs d'entreprise tournent sur des systèmes comme Solaris, AIX, HP-UX, etc., plutôt que sur Linux.

Beaucoup de développeurs Linux ont remarqué ce problème et essayé de le résoudre. Dans les versions expérimentales 2.3.x et la version 2.4 cette limitation a été traitée [...].

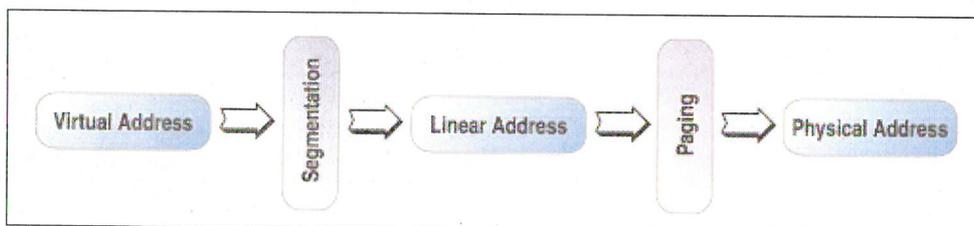
Cela veut-il dire que nous devons choisir un autre système d'exploitation ? Est-ce possible de trouver une solution qui permette de dépasser cette limitation des noyaux 2.2.x ? Afin de répondre à cette question nous devons d'abord savoir comment la gestion des processus s'effectue avec le noyau 2.2.x.

L'architecture i386 et la gestion de la mémoire sous Linux 2.2.x

La gestion des processus est intimement liée avec la gestion de la mémoire. Puisque l'implémentation de la gestion de la mémoire est basée sur l'architecture du matériel, nous devons nous intéresser en premier lieu à l'architecture i386. Dans les systèmes d'exploitation modernes, la technologie de la mémoire virtuelle est largement employée. Grâce à elle, les logiciels peuvent utiliser plus de mémoire que ce qui est physiquement présent. Cela signifie que les adresses mémoires utilisées par les applications sont virtuelles et converties en adresses réelles lors des accès par un mécanisme propre au processeur.

Il y a deux méthodes principales de gestion de la mémoire : la segmentation et la pagination. La segmentation consiste en la division de la mémoire en plusieurs segments où les accès mémoire se font avec un pointeur de segment et un *offset*. Cette méthode est utilisée par les systèmes anciens tels que PDP-11, etc. La pagination consiste en la division de la mémoire en plusieurs pages de taille fixe et en l'utilisation des pages comme unités de base de la gestion mémoire. Lors des accès mémoire, une adresse est convertie en une adresse physique en accord avec la table des pages.

La gestion de la mémoire avec l'architecture i386 est appelée segmentation avec pagination. L'espace d'adressage virtuel est d'abord divisé en segments au moyen de deux tables : la *Global Descriptor Table* (GDT) et la



> Figure 1 : la conversion d'une adresse virtuelle.

Local Descriptor Table (LDT). Après cela, l'adresse virtuelle est convertie en une adresse linéaire. Enfin, celle-ci est convertie en adresse physique au moyen de deux autres tables : la table de répertoire des pages (*Page Directory Table*) et la table des pages. La **figure 1** montre comment une adresse virtuelle est convertie en adresse réelle.

Sous Linux, le noyau tourne avec le privilège 0 (**ring 0**). En installant la GDT le noyau met son code et ses données dans des espaces d'adressage séparés. Tous les autres programmes tournent avec le privilège 3 (**ring 3**) avec leurs données et leur code dans le même espace d'adressage. La création de tables de pages différentes a pour but de protéger les tables des programmes utilisateurs. La table GDT sous Linux 2.2.x est montrée **figure 2**. En pratique, un programme utilisateur peut utiliser d'autres segments code/données en modifiant la LDT.

...
...
APM BIOS Data
APM BIOS 16-Bit Code
APM BIOS Code
APM BIOS Reserve
Reserve
Reserve
User Data
User Code
Kernel Data
Kernel Code
Unused Descriptor
NULL Descriptor

> Figure 2 : la table des descripteurs globaux (GDT = Global Descriptor Table).

La gestion des processus dans le noyau 2.2.x

Un processus est un programme qui tourne, ses ressources étant allouées. C'est un concept dynamique. Avec l'architecture i386, « tâche » est un autre nom pour un processus. Par commodité, nous n'emploierons que ce dernier terme. La gestion des processus est un concept qui concerne l'initialisation du système, leur création et leur destruction, la planification, la communication inter-processus, etc. Sous Linux, un processus est un groupe de structures de données incluant le contexte du processus, les données de planification, les sémaphores, la mise en file d'attente, l'identificateur du processus, le temps, les signaux, etc. Ce groupe est appelé *Process Control Block* ou PCB. Dans l'implémentation, le PCB est au bas de la pile du processus.

La gestion des processus Linux repose grandement sur l'architecture matérielle. Nous avons seulement parlé des bases de la gestion de mémoire segment-page du i386, mais en fait, un segment a un rôle plus important que d'être un simple bloc de mémoire. Par exemple, le *Task Status Segment* est un des plus importants segments du i386. Il contient beaucoup des données requises par le système. Chaque processus doit avoir un TSS pointé par le registre TR. En accord avec la définition du i386, le sélecteur dans TR doit sélectionner un descripteur dans la GDT. De surcroît, le sélecteur dans LDTR, qui définit la LDT d'un processus, doit avoir une entrée correspondante dans la GDT.

Afin de satisfaire toutes les conditions énumérées ci-dessus, la GDT de Linux 2.2.x est allouée pour tous les processus possibles. Le nombre maximum de processus simultanés est défini quand le noyau démarre. Le noyau réserve deux entrées GDT pour chaque processus.

Initialisation du système

Avec Linux 2.2.x quelques structures concernées par la gestion des processus sont initialisées au démarrage du système. La plus importante de celles-ci est la GDT et la liste des processus.

Quand le noyau démarre, il doit décider de la taille de la GDT. Puisque pour chaque processus, il faut deux entrées dans la GDT, la taille de cette dernière est définie par le nombre maximum de processus simultanés. Avec Linux, ce nombre est défini comme `NR_TASKS` au moment de la compilation. Comme le montre la **figure 2**, la taille de la GDT est de `10+2(avec APM)+NR_TASKS*2`. La liste des processus est un tableau de pointeurs PCB et est définie comme ceci :

```
Struct task_struct F= {&init_task,};
```

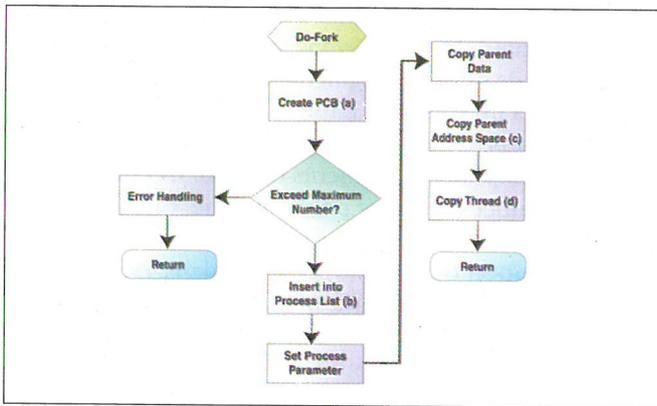
Dans cette ligne, `init_task` est le PCB du processus *root*. Après avoir inséré ce processus dans la liste des processus, le mécanisme de gestion des processus peut commencer à travailler. Notez que la taille de la liste des processus est, elle aussi, dépendante de `NR_TASKS`.

Création des processus

Sous Linux 2.2.x un processus est créé par un appel système : `fork`. Le nouveau processus est l'enfant du premier. Utiliser un clone permet de créer un *thread* qui est un processus léger. En fait, il n'y a pas vraiment de *thread* dans Linux 2.2.x. La **figure 3** montre comment l'appel système `fork` travaille.

Les étapes clé de `fork` sont les suivantes :

- Création du PCB du nouveau processus : le noyau alloue deux pages pour la pile du nouveau processus et le PCB est placé en bas de celle-ci.
- Insertion du processus dans la liste des processus : le noyau doit trouver une entrée libre dans cette liste. Si le système a atteint le nombre maximum de processus simultanés, il n'y a pas d'entrée libre dans la liste et l'appel système `fork` échoue.



> Figure 3: l'appel système fork.

- Copie de l'espace d'adressage du parent : le processus enfant a son espace d'adressage propre, mais d'abord, il partage l'espace d'adressage avec son parent au moyen d'un mécanisme « *copy-on-write* ». Le descripteur GDT du nouveau processus est également créé lors de cette étape.
- Initialisation du TSS pour le nouveau processus : le TSS du nouveau processus est créé dans le PCB et le descripteur GDT correspondant est également créé.

Planification des tâches

L'algorithme de planification (*scheduling*) est listé ci-dessous. Mais ici, nous ne regardons que la commutation des tâches. Sous Linux 2.2.x la commutation est réalisée par la fonction `switch_to`. Cela fonctionne comme ceci :

- chargement du nouveau TSS en initialisant TR;
- sauvegarde des anciens contenus des registres FS et GS dans l'ancien PCB;
- chargement de la LDT si cela est nécessaire pour le nouveau processus;
- chargement de la nouvelle table de pages pour le nouveau processus;
- chargement de FS et GS pour le nouveau processus.

Notez que les valeurs de TR et LDTR proviennent du PCB.

Dépasser le nombre de processus maximum

Qu'est-ce que le nombre de processus maximum ? En nous référant à ce que nous avons dit, il est facile de comprendre pourquoi il y a cette restriction sur le nombre de processus maximum. Au moment de la compilation, la valeur `NR_TASKS` définit statiquement le nombre maximum de processus simultanés, ainsi que la taille de la GDT dont le maximum est $8192 * 8$ octets, comme le définit l'architecture i386, ce qui signifie que la table peut contenir 8192 descripteurs. Quand un noyau 2.2.x démarre, la GDT est usitée comme ceci :

- descripteur NULL (entrée 0), descripteurs réservés (entrées 1,6,7);
- descripteurs de code et données du noyau (entrées 2,3) et descripteurs de code et données utilisateur (entrées 4,5);
- descripteur APM BIOS (entrées 8-11).

Au total, 12 entrées sont utilisées. Et puisque chaque processus a besoin de deux entrées dans la GDT, le nombre théorique de processus simultanés est $(8192-12)/2 = 4090$.

La solution au problème

Bien que la taille de la GDT soit limitée par le matériel, nous pouvons trouver une solution pour ce problème. Pour un CPU seul, un processus peut s'exécuter à un moment donné. Cela veut dire qu'il n'est pas nécessaire de réserver les descripteurs GDT pour tous les processus possibles. Quand un processus est sur le point de tourner, nous initialisons les descripteurs dynamiquement.

Après analyse de la structure PCB, nous pouvons y trouver le TSS et la LDT (le cas échéant). Ainsi, au cours d'une commutation de processus nous pouvons trouver ces deux segments à travers le pointeur PCB comme ceci :

```
TSS: proc->tss
LDT:proc->mm->segments
```

En fait, au cours de la commutation de processus nous pouvons trouver le pointeur PCB dans la liste des processus. Puisque TSS et LDT peuvent tous deux être obtenus de cette façon, il n'est pas nécessaire de les conserver en permanence dans la GDT.

Notre solution est de réserver seulement deux descripteurs GDT pour chaque CPU et de conserver le reste pour tous les processus. Par exemple, sur une machine équipée de deux CPU, quatre entrées sont réservées dans la GDT. Quand un processus A est attribué au CPU 1, les entrées GDT trois et quatre sont affectées avec les descripteurs TSS et LDT du processus A. Les anciennes valeurs de ces entrées sont effacées. Les entrées restantes de la GDT sont utilisées exactement comme dans le système d'origine.

L'implémentation en bref

La base de notre solution consiste à définir dynamiquement les descripteurs TSS et LDT de notre processus (voir le [listing 1](#)).

Listing 1. Initialisation du système

```
# définir la taille de la GDT à sa valeur maximum de 8192
# dans head.S
# Dans la définition de la GDT dans le fichier desc.h, insérer
# les entrées réservées pour chaque CPU et réserver toutes
# les autres entrées pour les processus
# de la même façon que le système original.
```

```
CPU0: SHARED_TSS_ENTRY = 12
SHARED_LDT_ENTRY = 13
CPU1: SHARED_TSS_ENTRY + 1 = 14
SHARED_LDT_ENTRY + 1 = 15
...
CPUn: SHARED_TSS_ENTRY + n = SHARED_TSS_ENTRY + n
SHARED_LDT_ENTRY + n = SHARED_LDT_ENTRY + n
( n < NR_CPUS )
```

```
# Changer NR_TASKS d'une macro en une variable, et
# faire ceci dans la fonction d'initialisation
```

```
# du système start_kernel():
```

```
Task = kmalloc(sizeof(void *) * NR_TASKS,
               GFP_ATOMIC);
# Ceci va allouer dynamiquement la liste des
# processus
# Modifier la fonction parse_options pour accepter
# un paramètre supplémentaire nrtasks, qui indique le
# nombre maximum de processus simultanés. Ceci permet à
# l'utilisateur de définir dynamiquement le nombre de
# maximum.
```

La commutation de processus

Dans la version d'origine, lors d'une opération *fork* les `tss.ldt` et `tss.tr` dans PCB sont employés pour sauvegarder les sélecteurs dans LDTR et TR. Selon l'algorithme d'origine, le sélecteur LDT d'un processus peut dépasser sa limite de 16 bits. Ainsi, nous employons une variable supplémentaire `tss._ldth` conjointement à `tss.ldt` pour sauvegarder le sélecteur. Puisque `tss._ldth` n'est pas utilisé dans Linux 2.2.x, notre modification ne détruit pas le noyau. La sauvegarde de LDTR et TR fonctionne maintenant comme ceci :

```
((unsigned long *) & (p->tss.ldt)) =
(unsigned long)_LDT(nr);
if (*(unsigned long *) & (p->tss.ldt) <
(unsigned long)(8192 << 3)
set_ldt_desc(nr, ldt, LDT_ENTRIES);
// code original ici sinon{
// ne rien faire
// laisser le code processus de commutation manipuler LDT
// and TSS
}
```

Un des bénéfices de cette implémentation est que nous pouvons facilement découvrir si le numéro de processus est supérieur à 4088 en inspectant la valeur de `tss.ldt`. Ceci est important pour les performances.

Si le numéro de processus est supérieur à 4088 celui-ci n'a pas de descripteur réservé dans la GDT et il doit en utiliser les entrées partagées. Nous pouvons trouver ces entrées avec ce code :

```
SHARED_TSS_ENTRY + smp_processor_id()
```

Le **listing 2** montre le code s'occupant des entrées partagées de la GDT.

Listing 2. Utilisation des entrées partagées de la GDT

```
...
#define set_shared_tss_desc(addr,cpu)\
_set_tssldt_desc(gdt_table+
                SHARED_TSS_ENTRY+2*cpu,(int)addr,235,0x89);
#define set_shared_ldt_desc(addr,size,cpu)\
_set_tssldt_desc(gdt_table+SHARED_LDT_ENTRY+2*cpu,
                (int)addr,((size<3)-1),0x82);
...
void __switch_to(task_struct *prev,
                task_struct *next){
...
if(next->tss.tr <= 0x0000ffff)
```

```
{
    //original code here
} else {
    set_shared_tss_desc(&next->tss),
    smp_processor_id());
    set_shared_ldt_desc(&next->mm->segments,
    LDT_ENTRIES,smp_processor_id());
}
//set LDTR and TR
...
}
```

Après avoir fait ceci, nous dépassons la restriction sur le nombre maximum de processus. Nous pouvons même ajouter un paramètre supplémentaire dans le fichier de configuration de `lilo` pour définir ce nombre dynamiquement. La ligne suivante porte la limite à 40 000 ce qui est beaucoup plus que 4090 :

```
Append = "nrtasks=40000"
```

Pour conclure

Grâce à cette solution nous pouvons définir le nombre limite de processus à 2 Go. En théorie. Mais en pratique, le matériel et le système d'exploitation limitent ce nombre. Lorsqu'il crée un nouveau processus le noyau lui alloue de la mémoire comme ceci :

```
Process stack (2 pages) + page table (1 page)
+ page directory table (1 page) = 4 pages
```

Ainsi, si l'ordinateur dispose d'1 Go de mémoire et utilise cinq pages par processus et que le système requiert 20 Mo de mémoire, le nombre maximum de processus peut être :

$(1 \text{ Go} - 20 \text{ Mo}) / 20 \text{ Ko} = 51404 \sim 50000$

Plus pratiquement un processus utilisera 30 Ko de mémoire au moins, ainsi le nombre limite est maintenant :

$50000 * (2/3) = 33000$

Ce nombre est encore beaucoup plus grand que 4090. ■

L'auteur :

Zhang Yong (leon@xteamlinux.com.cn) est ingénieur informatique à Xteam Software Co., Ltd.

Son travail couvre de nombreux aspects de Linux, y compris le développement du noyau, Linux 18N&110N et les applications réseau, etc.

Il est actuellement concentré sur la nouvelle version sur serveur XteamServer qui est une solution serveur haut de gamme basée sur Linux.

Xteam Software Co., Ltd. est aussi vendeur de XteamLinux et XteamWindows. Ce sont les deux distributions les plus populaires en Chine. Pour plus d'informations visitez <http://www.xteamlinux.com.cn/>.

Initiation à OpenGL fin

Pour terminer notre initiation à OpenGL nous allons nous intéresser aux fontes, aux bitmaps et aux images.

Frédéric Mazué

fredericmazue@wanadoo.fr

Sur votre CD-Rom

Les fontes, bitmaps et images sont des objets purement 2D. À première vue, il peut sembler assez surprenant qu'une bibliothèque comme OpenGL, normalement dédiée à l'écriture d'applications 3D, propose aussi ce type de fonctionnalités. Pourtant les objets 2D sont souvent présents dans les applications 3D. Par exemple, un jeu qui affiche un tableau de scores aura besoin de fonctionnalités liées aux fontes. Les bitmaps peuvent servir à la définition de fontes personnalisées et les images sont à la base des textures. Un jeu comme Tomb Raider utilise abondamment les textures pour définir les aspects des parois des labyrinthes, des rochers, etc. Il est vrai que Tomb Raider n'est pas écrit avec OpenGL mais avec Direct3D de Microsoft. Le principe reste néanmoins le même. Pour prendre un autre exemple qui fera plaisir aux linuxiens, Quake 3, écrit avec Mesa, l'implémentation d'OpenGL sous Linux, fait abondamment usage d'images appliquées en textures pour la création des décors. Nous voyons donc que les éléments 2D ont une place importante dans les applications 3D. Nous ne parlerons pas des textures dans cet article car ce sujet nous entraînerait beaucoup trop loin. Nous nous limiterons à aborder les images qui sont à la base des textures.

Les fontes

Il y a deux méthodes pour afficher des textes à l'écran avec OpenGL. La première méthode est celle des paresseux. Au début de cette initiation nous avons dit qu'OpenGL est une bibliothèque de bas niveau. Toutes les bibliothèques 3D sont de bas niveau pour des questions évidentes de performances. Le revers de la médaille est que le programmeur peut avoir un surcroît de travail pour certaines tâches. Afficher des textes est l'une de ces tâches. Nous avons également dit qu'OpenGL est accompagné de deux bibliothèques

de plus haut niveau visant à soulager le développeur. Ce sont GLU et glut. La bibliothèque glut fournit des fonctionnalités d'affichage de textes à l'attention des paresseux.

Lorsque une application OpenGL s'appuyant sur glut démarre, 7 fontes de types bitmap (nous reviendrons sur la signification de cette appellation un peu plus loin) sont chargées, et des pointeurs sur ces fontes sont conservés par glut afin que le programmeur puisse y accéder facilement. Ce sont des fontes natives du système d'exploitation hôte. Ceci implique que des applications Windows et Linux les utilisant auront des aspects voisins mais différents, puisque les fontes disponibles sur ces systèmes sont légèrement différentes. Dans le fichier en-tête `glut.h` figurent des macros définissant les noms de ces fontes tout en permettant d'accéder aux pointeurs internes. Les fontes sont de deux types : proportionnelles et à pas fixe. Voici leurs noms accompagnés d'une brève description :

- **GLUT_BITMAP_8_BY_13** : chaque caractère tient dans un rectangle de 8*13 pixels. Il s'agit donc d'une fonte à pas fixe. À l'initialisation, glut charge la fonte « courier » sous Windows et la fonte « fixed » sous Linux.
- **GLUT_BITMAP_9_BY_15** : c'est la même fonte que précédemment mais avec des caractères plus gros : 9*15 pixels.
- **GLUT_BITMAP_TIMES_ROMAN_10** : fonte proportionnelle 10 points. À l'initialisation glut charge la fonte « times

new roman» sous Windows et la fonte «times» sous Linux.

- **GLUT_BITMAP_TIMES_ROMAN_24** : même fonte que précédemment, mais à 24 points cette fois.
- **GLUT_BITMAP_HELVETICA_10** : fonte proportionnelle 10 points. À l'initialisation glut charge la fonte «Arial» sous Windows et «helvetica» sous Linux.
- **GLUT_BITMAP_HELVETICA_12** : même fonte que précédemment, mais à 12 points.
- **GLUT_BITMAP_HELVETICA_18** : même fonte, mais à 18 points.

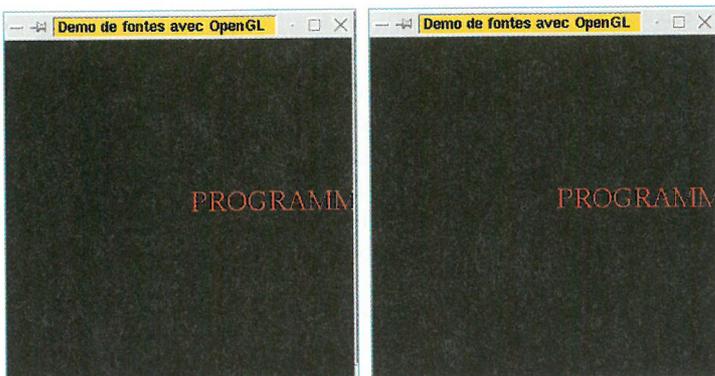
Positionner et afficher un caractère

Avant d'afficher quoi que ce soit, il faut indiquer à OpenGL l'endroit de l'écran où effectuer l'opération. Dans le premier article de cette série, nous avons appris qu'OpenGL utilise deux systèmes de coordonnées : un système de coordonnées de *viewport* qui d'un point de vue pratique sont les coordonnées en pixels dans la fenêtre et un système de coordonnées dépendant des matrices de projection. Ce dernier système étant utilisé pour rendre les objets 3D.

Les fontes sont des objets 2D. Avec OpenGL, les objets 2D sont appelés des trames, et les coordonnées de trames ne sont pas modifiées par la matrice de projection. Pour restituer un objet trame à un emplacement déterminé, il faut donc spécifier les coordonnées de l'emplacement de tramage en coordonnées fenêtre. Le point origine (0, 0) se situe en bas et à gauche de la fenêtre. De même, le point de tramage spécifié se situe en bas et à gauche du caractère devant être dessiné. Prenons un exemple :

```
glColor3f(1.0, 0.0, 0.0);
glRasterPos2d(100, 100);
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'A');
```

Ce morceau de code commence par spécifier la couleur dans laquelle sera affichée le caractère. Le rouge dans ce cas. Ensuite, la trame est positionnée aux coordonnées (100, 100, l'unité est le pixel) à partir du coin inférieur gauche de la fenêtre. Enfin le caractère «A» est affiché. Il apparaîtra en haut et à droite du point de coordonnées (100, 100).



> Figures 1 et 2 : un scrolling de texte avec OpenGL et glut.

Quelques remarques et quelques pièges à éviter

- La fonction `glutBitmapCharacter` est prévue pour afficher un seul caractère à la fois. Elle se charge d'appeler la fonction `glBitmap` d'OpenGL de telle façon qu'après l'affichage, la position de la trame (ou *raster*) soit déplacée vers le droite de la largeur du caractère plus un pixel. Il est donc possible d'afficher des chaînes de caractères en écrivant une boucle autour de `glutBitmapCharacter`.
- Veillez à sélectionner la couleur de l'affichage AVANT d'appeler la fonction `glRasterPos`, sinon la couleur risque de ne pas être sélectionnée comme vous le souhaitez. Par exemple :

```
glColor3f(1.0, 0.0, 0.0); // couleur rouge
glRasterPos2d(100, 100);
glColor3f(0.0, 1.0, 0.0); // couleur verte
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'A');
```

ne marche pas. Le caractère affiché sera de couleur rouge. Le second appel à `glColor` n'est pas pris en compte.

- La position du *raster* doit être valide pour que l'affichage soit réellement effectué. Si sa position se trouve en dehors de la fenêtre, rien n'apparaîtra à l'écran, même si la taille de l'objet à représenter pouvait faire espérer le contraire. Autrement dit, il est important de ne pas faire la confusion entre la position du *raster* et les opérations de *clipping*. Si une position de *raster* est valide et que l'objet à représenter «déborde» de la fenêtre, OpenGL fera automatiquement un *clipping*, c'est-à-dire supprimera les parties de l'objet qui ne peuvent être représentées. Si la position du *raster* n'est pas valide, RIEN n'est représenté.

Le **listing partiel 1** met en évidence les première et troisième remarques ci-dessus. Il s'agit du *scrolling* d'une chaîne de caractères (voir les **figures 1 et 2**).

Essayez ce programme. Vous constaterez que le texte est coupé (ou clippé) sur le côté droit de la fenêtre et n'est pas affiché lorsque la position du *raster* sort de la fenêtre à gauche.

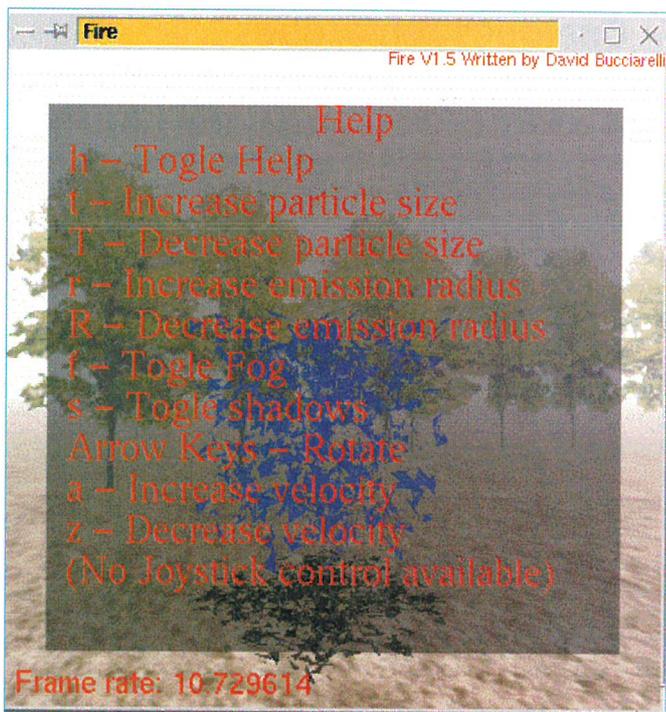
```
// Listing partiel 1
//
// Cette fonction d'affichage génère un scrolling
// de texte à l'écran
//

#define WIDTH 300
#define HEIGHT 300

char logo[] = "PROGRAMMEZ!";

GLfloat couleurs[3][3]=
{{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}};

void draw(void)
{
    static int x = WIDTH;
    const int y = 150;
```



> Figure 3 : le programme de démonstration «fire» de Mesa utilise abondamment la fonction `glutBitmapCharacter`.

```

glClear(GL_COLOR_BUFFER_BIT);

glColor3fv(couleurs[0]);
glRasterPos2d(x, y);
//glColor3fv(couleurs[1]);
//glPixelZoom(2.0, 2.0);
for(int i=0; i<strlen(logo); i++)
    glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, (int)logo[i]);
x--;

// En fait le texte n'est plus affiché dès que x vaut 0
// car la position du raster devient invalide.
if(x<0-WIDTH)
{
    x = WIDTH;
}
glutSwapBuffers();
}

static void idle(void)
{
    glutPostRedisplay();
}

```

Ainsi que nous l'avons dit, cette méthode est la plus simple puisque nous pouvons utiliser des fontes préfabriquées grâce à glut qui est une bibliothèque de plus haut niveau qu'OpenGL. Les programmes de démonstration qui accompagnent Mesa, la version *Open Source* d'OpenGL sous Linux, utilisent abondamment cette méthode, avec des résultats tout-à-fait honorables (voir la [figure 3](#)).

Les fontes bitmap

Si vous n'êtes pas satisfait des fontes proposées par glut, il faudra vous retrousser les manches et créer vos propres

fontes. En fait les caractères n'existent pas au niveau d'OpenGL. Un caractère n'est rien d'autre qu'une bitmap. Lorsque glut charge les fontes depuis le système hôte, elle crée, à partir de ces fontes, des jeux de bitmaps qui serviront à la représentation des caractères.

La fonction `glutBitmapCharacter` appelle simplement la fonction OpenGL `glBitmap` pour tracer la bitmap correspondante au caractère devant être «écrit». L'emploi des bitmaps n'est évidemment pas limité au tracé de caractères.

Le terme de bitmap ne doit pas induire en erreur. Il ne faut pas confondre les bitmaps OpenGL avec les bitmaps Microsoft Windows. Les bitmaps OpenGL sont des champs de bits au sens strict. C'est-à-dire que chaque bit correspond à un pixel. La bitmap ne contient donc aucune information de couleur relative aux pixels. Ainsi, tous les pixels d'une bitmap sont affichés en une seule et même couleur : la couleur sélectionnée par la fonction `glColor`.

Les bitmaps Windows contiennent, en plus des informations de pixels, des informations relatives aux couleurs de ceux-ci. En cela, les bitmaps Windows s'apparentent aux images OpenGL. Nous y reviendrons.

Il est facile, quoique fastidieux, de créer des fontes personnalisées sous OpenGL. Il faut et il suffit de définir une bitmap pour chaque caractère. Par contre, nous pouvons donner libre cours à notre imagination en ce qui concerne la taille et la forme des lettres. Il n'est même pas obligatoire que les lettres aient toutes la même taille.

Supposons que nous voulions créer une bitmap pour représenter la lettre A dans un carré de 8*8 pixels. Il nous suffit donc de définir 8 octets. Les bits à 0 de ces octets correspondront à la couleur du fond et les bits à 1 correspondront à la couleur active de l'affichage. Voici une telle bitmap, le caractère 0 correspondant aux bits à 0 et le caractère * correspondant aux bits à 1. En face de chaque ligne apparaît la valeur hexadécimale de chaque octet :

```

ooo**ooo      0x18
oo****oo      0x3C
oo*oo*oo      0x24
o*oooo*o      0x42
o*oooo*o      0x42
*****        0xFF
*ooooo*       0x81
*ooooo*       0x81

```

Nous avons vu que le *raster* se positionne en bas et à gauche de la bitmap-caractère à représenter. En interne, cela implique qu'OpenGL trace le caractère «en montant», c'est-à-dire en commençant par la ligne du bas pour terminer par la ligne du haut. Il faut donc définir la bitmap dans cet ordre lorsqu'on passe à la programmation :

```

Glubyte monchar[] = {0x81, 0x81, 0xFF, 0x42, 0x42, 0x24, 0x3c, 0x18};

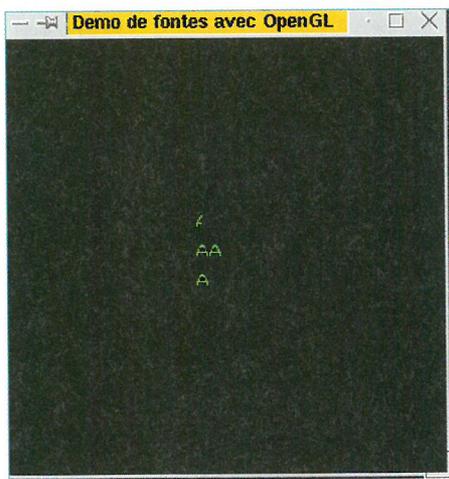
```

L'opération pour tracer un caractère à l'écran est tout-à-fait semblable à ce que l'on a fait avec glut :

```
glColor3f(0.0, 1.0, 0.0);
glRasterPos2d(100, 100);
glBitmap(8, 8, 0, 0, 9, 0, monchar);
```

Les paramètres reçus par la fonction `glBitmap` méritent toute notre attention.

- Les deux premiers paramètres spécifient la largeur et la hauteur de la bitmap à tracer. Ces valeurs ne doivent pas forcément correspondre à la taille réelle de la bitmap. S'il y a correspondance, la bitmap est représentée normalement. Si la valeur est inférieure à la taille de la bitmap, celle-ci est représentée tronquée, mais sans déformation. Par contre, si la valeur est supérieure à la taille de la bitmap, on obtient un dessin bizarre dans le meilleur des cas et une faute de protection si on est malchanceux.
- Les deux paramètres suivants permettent de spécifier l'origine à l'intérieur de la bitmap. Autrement dit le *raster* ne se positionne plus en bas et à gauche de la bitmap, mais sur ce point nouvellement défini.
- Les cinquième et sixième paramètres définissent les valeurs d'incrément de la position du *raster* en x et y après le tracé. Une valeur en x inférieure à la taille de la bitmap fera se chevaucher les caractères. Une valeur non nulle en y permet d'obtenir des effets d'escalier. C'est un ajustement judicieux de ces paramètres qui nous permettront d'éviter des chevauchements désagréables dans le cas où les lettres de notre fonte n'auraient pas toutes la même taille.



> Figure 4: exemple de fontes personnalisée.

- Le dernier paramètre est un pointeur sur les octets de la bitmap. Pour que la fonction `glBitmap` puisse lire ces octets correctement, elle doit savoir comment ceux-ci sont rangés en mémoire. Dans l'exemple ci-dessus, ils sont rangés de manière traditionnelle, mais il est possible de procéder autrement. Pour spécifier comment les octets sont rangés en mémoire, il faut utiliser la fonction `glPixelStore`. L'usage classique est `glPixelStorei(GL_UNPACK_ALIGNMENT, 1)`. Le lecteur voudra bien se reporter à la documentation OpenGL en cas de besoin.

Le **listing partiel 2** trace notre caractère «A» d'abord une fois, puis deux fois successivement comme dans le cas d'une chaîne de caractères, puis trace un caractère tronqué (voir la **figure 4**).

Petite remarque : par commodité l'appel à la fonction `glPixelStore` figure dans la fonction de tracé. En fait, pour des raisons de performances, il est plus judicieux de faire cet appel une fois pour toutes dans une fonction d'initialisation.

```
// Listing partiel 2
GLfloat couleurs[3][3]={{1.0, 0.0, 0.0},
{0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}};

GLubyte monchar[] = {0x81, 0x81, 0xFF,
0x42, 0x42, 0x24, 0x3c, 0x18};

void draw(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glColor3fv(couleurs[1]);
    // Affichage d'un simple caractère
    glRasterPos2d(130, 130);
    glBitmap(8, 8, 0, 0, 9, 0, monchar);
    // Affichage de deux caractères successivement
    glRasterPos2d(130, 150);
    glBitmap(8, 8, 0, 0, 9, 0, monchar);
    glBitmap(8, 8, 0, 0, 9, 0, monchar);
    // Affichage d'un 'demi-caractère
    glRasterPos2d(130, 170);
    glBitmap(4, 8, 0, 0, 9, 0, monchar);

    glutSwapBuffers();
}
```

Les images

Le sujet, cette fois, est plus ardu. Les formats d'image OpenGL sont nombreux. Ils correspondent à la façon dont sont stockées les informations en mémoire. Tant que l'on se limite à conserver des informations de couleurs non compressées, les choses restent simples. Si l'on souhaite ajouter des composantes alpha, des composantes de luminances ou de profondeur, le tout étant compressé pour économiser l'espace mémoire, la manipulation des images OpenGL se complique très rapidement. De plus, se pose le problème de la persistance des informations. En tant que bibliothèque de bas niveau, OpenGL n'offre pas de facilités d'entrées/sorties permettant de sauve-

garder une image dans un fichier pour la recharger plus tard en mémoire. L'écriture de telles fonctions reste à la charge du programmeur.

OpenGL met principalement quatre fonctions pour manipuler les images dans la mémoire de la machine et les tracer à l'écran.

- `glDrawPixels` trace une image à l'écran à partir d'un tableau de pixels en mémoire centrale.
- `glReadPixels` effectue l'opération inverse. Elle récupère les pixels affichés à l'écran (ou plus exactement dans la mémoire graphique) et place les informations en mémoire centrale.
- `glCopyPixels` copie une image (ou une portion d'image à l'écran) pour la restituer à un autre endroit de l'écran. On peut obtenir la même chose avec des appels successifs à `glReadPixels` et `glDrawPixels`. Par contre, `glCopyPixels` est plus rapide car les informations ne transitent pas par la mémoire centrale de la machine.
- `glPixelZoom` permet d'agrandir ou réduire une image. Elle permet encore d'obtenir des effets miroir.

// Listing partiel 3

```
#define ImageWidth 64
#define ImageHeight 64

GLubyte Image[ImageHeight][ImageWidth][3];

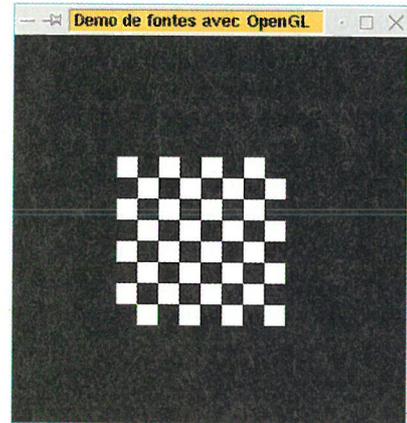
void BuildImage()
{
    int i, j, c;

    for(i=0; i<ImageHeight; i++)
    {
        for(j=0; j<ImageWidth; j++)
        {
            c = (((i&0x08)==0)^(j&0x08)==0)*255;
            Image[i][j][0] = (GLubyte)c;
            Image[i][j][1] = (GLubyte)c;
            Image[i][j][2] = (GLubyte)c;
        }
    }
}

void draw(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glRasterPos2d(80, 80);
    glPixelZoom(2.0, 2.0);
    glDrawPixels(ImageWidth, ImageHeight, GL_RGB, GL_UNSIGNED_BYTE, Image);
    glutSwapBuffers();
}

static void init(int argc, char *argv[])
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_FLAT);
    BuildImage();
}
```



> Figure 5: ce que vous obtenez avec le listing 3.

Le plus simple, pour se faire la main avec les images est de travailler avec le format qui est adopté par les bitmaps 24 bits sous Windows. Dans ce format, chaque pixel est défini par trois octets (soit 24 bits). La valeur du premier octet correspond à la composante rouge du pixel, la valeur du deuxième octet à la composante verte, et la troisième valeur à la composante bleu.

Pour donner un exemple, définissons une bitmap de 3 pixels et composée d'un point rouge, d'un point vert et d'un point bleu. Notre image sera donc définie par 9 octets au total :

```
GLubyte mbitmap[3][3] = {
    {0xFF, 0x00, 0x00}, // point de couleur rouge
    {0x00, 0xFF, 0x00}, // point de couleur verte
    {0x00, 0x00, 0xFF} // point de couleur bleu
};
```

Le **listing partiel 3** construit, dans le format expliqué plus haut, une image qui représente un damier. Le damier est agrandi lors de l'affichage au moyen de la fonction `glPixelZoom` (voir la **figure 5**). La fonction qui construit l'image a été prise dans l'ouvrage mentionné plus bas.

La fin d'une initiation fait naître le désir de perfectionnement

Notre initiation à OpenGL se termine mais nous sommes très loin d'avoir épuisé le sujet. Nous souhaitons seulement avoir donné le goût de l'infographie 3D au lecteur. Si celui-ci souhaite aller plus loin, nous lui recommandons la lecture d'un ouvrage de référence comme: *OpenGL 1.2 Guide Officiel*, Mason Woo, Jackie Neider, Tom Davis Dave Shreiner, Éditions CampuPress. Il y trouvera les informations relatives aux différents formats d'images, aux textures, au calcul de normales pour l'éclairage, au lissage, aux surfaces gauches, etc... Bref, tout ce qu'il faut pour écrire des applications 3D de qualité professionnelle. Bon courage! ■

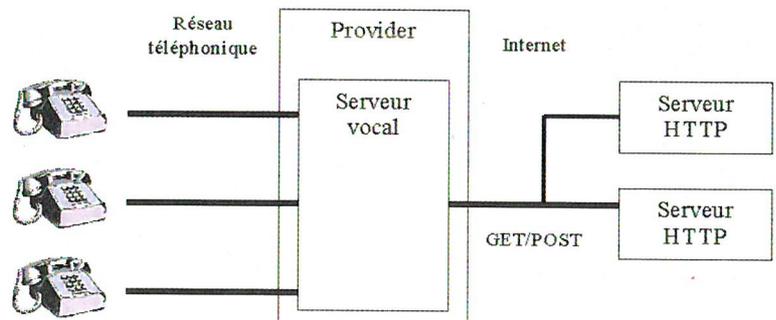
Faites-vous entendre grâce à VoiceXML

VoiceXML permet d'offrir facilement une application vocale à partir d'un site Internet. Est-ce à dire que le «phone» est l'avenir du «nerd»?

Philippe PRADOS
philippe@prados-obj.nom.fr

En 2005, la majorité des internautes utiliseront un autre périphérique qu'un ordinateur pour naviguer sur le réseau.

Le WAP est une technologie permettant de consulter son compte, de passer des ordres en Bourse, ou de jouer avec son téléphone. Pour le développeur, l'hétérogénéité des téléphones et les bugs des implantations rend très difficile la réalisation d'une application WAP qui fonctionne. Pour le format HTML, il faut garantir une compatibilité entre Internet Explorer et Netscape Communicator, et éventuellement avec le navigateur Opéra.



Il existe pourtant un média universel, disponible dans tous les foyers, et sans problème de compatibilité : le téléphone. Le téléphone classique avec son micro et son haut-parleur est d'un accès plus facile. Cela permet de toucher pratiquement toute la population.

Une technologie permet de naviguer sur le Net avec un téléphone classique. Elle s'appelle VoiceXML. Que vient faire XML, qui est un format texte, avec le téléphone qui utilise des sons ?

VoiceXML permet de décrire l'ergonomie d'un site vocal. Un serveur particulier interprète le format VoiceXML pour diffuser les messages à l'utilisateur et pour analyser ses réponses au format DTMF (les touches du téléphone) ou par reconnaissance vocale. IBM possède 95 brevets sur la reconnaissance de la voix.

(<http://www.research.ibm.com/hlt/html/patents.html>)

Pour naviguer sur Internet, un programme analyse le format HTML, présente le document à l'utilisateur et interprète ses réactions pour remplir des formulaires. Un navigateur VoiceXML procède de même. Il demande initialement un document à un serveur http, diffuse vocalement les informations en synthèse vocale ou en jouant des fichiers pré-enregistrés ; puis, le navigateur analyse les réponses de l'utilisateur ou les touches du téléphone. Cela permet de remplir des formulaires pour demander les pages suivantes.

Comment le téléphone peut-il devenir un navigateur VoiceXML ? Il passe par un serveur intermédiaire s'occupant de faire le lien entre le téléphone et le Net. (Voir schéma) L'utilisateur appelle un numéro de téléphone. Le serveur vocal est paramétré pour associer un numéro à une URL. Il interroge alors le serveur http correspondant pour obtenir un fichier au format VoiceXML. Le serveur vocal peut alors proposer une ergonomie adaptée au téléphone et invoquer le

serveur http lorsque cela est nécessaire.

Ainsi, sur le serveur http, la même application peut être présente dans les formats HTML, WAP et VoiceXML. Cela permet de s'adapter au média de l'utilisateur. Le client peut choisir le média qu'il désire pour consulter son compte en banque ou passer des ordres en Bourse. À terme, l'utilisateur pourra utiliser simultanément les fonctionnalités WAP et VoiceXML. Le protocole WAP s'occupe des interactions textuelles, VoiceXML de lire ou d'enrichir ces informations. Les écrans des téléphones WAP sont insuffisamment spacieux pour une longue explication. Un message vocal pourra alors commenter les informations textuelles. Inversement, un tableau est très difficile à diffuser en vocal. Le WAP pourra alors prendre le relais. Réunir les différents médias sur le même serveur d'applications permettra d'unir ces technologies.

Les spécifications de VoiceXML sont disponibles sur le site officiel de référence www.voicexml.org. Elles ont été soumises au W3C.

Vous pouvez télécharger un navigateur VoiceXML en Java sur le site alphaworks.ibm.com. Cette version fonctionne dans deux configurations : sans interface vocale, avec simulation par le clavier des interactions de l'utilisateur, et une version vocale qui nécessite le logiciel ViaVoice d'IBM version US.

Nous allons regarder comment rédiger un fichier VoiceXML. Voici un premier fichier.

```
<?xml version="1.0"?>
<vxml version="1.0">
  <form>
    <block>Hello World!</block>
  </form>
</vxml>
```

Cet exemple demande au navigateur vocal de diffuser le message «Hello World!». Un moteur de synthèse analyse le texte contenu dans le marqueur **block** et diffuse le message en synthèse vocale.

Si le message est pré-enregistré par une voix chaleureuse, le fichier XML doit utiliser le marqueur **audio** et faire référence au fichier vocal correspondant.

```
<?xml version="1.0"?>
<vxml version="1.0">
  <form>
    <audio src="hello.wav"/>
  </form>
</vxml>
```

Deux modes sont proposés avec VoiceXML : les menus et les formulaires.

Les menus permettent de proposer à l'utilisateur différents chemins. Celui-ci traverse les liens à l'aide des touches du téléphone ou de mots-clés reconnus par le moteur de reconnaissance vocale.





```
<menu>
<prompt>Bonjour. Choisissez entre : <enumerate/></prompt>
<choice next="/sport/index.vxml">
  Sport
</choice>
<choice next="/meteo/index.vxml">
  Météo
</choice>
<choice next="/finance/index.vxml">
  Finance
</choice>
<noinput>
  Indiquez l'un des mots <enumerate/>
</noinput>
</menu>
```

Le marqueur **enumerate** diffuse les différents mots possibles du menu. Une session pourrait ressembler à ceci :

Serveur – Bonjour. Choisissez entre Sport, Météo, Finance.

Utilisateur : Bourse.

Serveur – Indiquez l'un des mots Sport, Météo, Finance.

Utilisateur : Météo.

le serveur diffuse l'information décrite dans le fichier /meteo/index.vxml.

Le deuxième mode proposé par VoiceXML permet à l'utilisateur d'entrer un formulaire. Une boucle principale s'exécute tant que tous les champs du formulaire ne sont pas renseignés correctement.

```
<form id="meteo">
<block>Bienvenu sur le service de météo.</block>
<field name="departement">
<prompt>Quel département ?</prompt>
<grammar src="departement.gram" type="application/x-jsgf"/>
<catch event="aide">
  Indiquez distinctement le nom d'un département.
</catch>
</field>
<field name="ville">
<prompt>Quelle villes ?</prompt>
<grammar src="ville.gram" type="application/x-jsgf"/>
<catch event="aide">
  Indiquez une ville du département.
</catch>
</field>
</block>
<submit next="meteo.jsp" namelist="departement ville"/>
</form>
```

Ce formulaire demande à l'utilisateur d'indiquer le département et la ville d'où il désire obtenir la météo. Le marqueur **grammar** permet d'indiquer un fichier contenant l'ensemble des mots du vocabulaire concerné par le champ.

Le marqueur **catch** permet de capturer des événements. En l'occurrence, l'exemple capture le mot « aide » pour diffuser un message particulier. Tant que l'ensemble des champs n'est pas renseigné, le navigateur VoiceXML ne soumet pas la requête. Une fois le formulaire entièrement validé, le marqueur **submit** s'occupe de demander la page **meteo.jsp** en lui indiquant les champs département et ville.

Techniquement, les formulaires donnent la possibilité de valoriser des variables JavaScript dont le nom est indiqué dans le marqueur **field**. Le marqueur **submit** indique la liste des variables à soumettre. Les variables JavaScript sont également manipulables par d'autres marqueurs VoiceXML.

```
<var name="un" expr="1"/>
<field name="deux" expr="un+1" type="number"/>
Cela permet de proposer une valeur calculée à un champ ou d'adapter le formulaire à l'aide d'if
<catch event="aide">
  <if cond="type_carte == 'amex'">
    Indiquez 15 chiffres.
  <else/>
    Indiquez 16 chiffres.
  </if>
</catch>
Dans un formulaire, l'utilisateur peut être amené à enregistrer un message.
<form>
  <record name="greeting" beep="true" maxtime="10s"
    finalsilence="4000ms" dtmfterm="true" type="audio/wav">
    <prompt>Après le bip, enregistrez votre message.</prompt>
    <noinput>Je n'ai rien entendu. Recommencer.</noinput>
  </record>
  ...
</form>
```

Le fichier correspondant est envoyé au serveur http lors de la soumission du formulaire.

Des scripts permettent d'enrichir les fonctionnalités de VoiceXML.

```
<?xml version="1.0"?>
<vxml version="1.0"?>
<script><![CDATA[
function factorial(n) { return (n <= 1)? 1 : n * factorial(n-1); }
]]> </script>
...
</vxml>
```

VoiceXML permet d'offrir facilement une application vocale à partir d'un site Internet. Pour que cela fonctionne, il faut faire appel à un *provider* qui s'occupera d'héberger le standard téléphone et le serveur vocal (DirectTalk d'IBM, par exemple). Le serveur http possédant l'application vocale peut se trouver n'importe où sur le réseau. À quand votre page vocale perso ? ■

Liens :

W3C : www.w3c.org

VoiceXML : www.voicexml.org

WML : www.allnetdevices.com/faq/

Divers : www.philippe.prados.net

Un cours accéléré en SDL

Adaptation d'un chapitre d'un livre de l'auteur, disponible sous peu et intitulé « Programming Linux Games ».

John Hall

Traduction : Xavier Leclercq
xavier.leclercq@programmez.com

Obtention de SDL

SDL est un logiciel gratuit (sous le LGPL) et est disponible pour téléchargement sur le site web <http://www.libsdl.org/>. En plus de la bibliothèque SDL actuelle, la page d'accueil de SDL fourmille d'exemples de code source, de démos, de jeux et d'extensions.

SDL est facile à installer à partir du package des sources mais la page d'accueil fournit aussi des binaires pour plusieurs des plates-formes les plus courantes.

Philosophie de conception de SDL

Si vous avez un jour travaillé avec DirectX de Microsoft, vous remarquerez que SDL est une bibliothèque minuscule en comparaison. Le code source de la *core library* ne fait qu'un peu moins de 6 mégaoctets, et cela inclut pas mal de code supplémentaire qui ne serait jamais lié à une application Linux. Cependant, ne vous y trompez pas, ces 6 Mo sont bien utilisés et la *core library* SDL fournit pratiquement tout ce dont vous avez besoin pour développer des jeux et des lecteurs média Linux de haute qualité. De plus, le site web accueille un certain nombre de bibliothèques supplémentaires offrant des caractéristiques additionnelles telles que le chargement d'images et le mixage audio de pointe. En maintenant ces caractéristiques séparées de la *core library*, SDL reste de petite taille et facile à apprendre.

La bibliothèque SDL consiste en plusieurs sub-API, offrant un support inter plates-formes pour la vidéo, l'audio, la gestion des entrées (*input handling*), le *multithreading*, les contextes de rendu OpenGL et autres éléments que les programmeurs de jeux apprécient. Malheureusement, nous ne disposons pas d'assez de place pour passer tout cela en revue et nous nous limiterons donc à la programmation vidéo et la gestion des entrées, les deux choses dont vous avez vraiment besoin pour se lancer dans SDL.

L'API vidéo de SDL

Le seul objectif de l'API vidéo de SDL est de trouver un appareil vidéo adéquat et de le configurer pour que votre applica-

Le secteur des jeux en Linux explose, en partie du fait que les gens aiment les jeux, mais aussi en raison des récents développements en multimédia Linux. Ces dernières années, quelques excellents outils multimédias orientés Linux sont apparus, tels que l'interface graphique GGI et le système de sons ALSA. La bibliothèque SDL a également fait quelque peu sensation récemment. SDL est une bibliothèque de programmation multimédia tout usage qui fournit une programmation portable et rapide aux graphiques, sons, périphériques d'entrée, *threads* et rendus OpenGL. La bibliothèque principale SDL, dite *core library*, est portable vers plusieurs versions d'UNIX ainsi que BeOS, MacOS et Win32. Ce qui en fait un choix excellent pour le développement de jeux sur plusieurs plates-formes sans compromettre les performances.

Contrairement à de nombreux outils multimédias, SDL ne communique pas réellement avec les appareils du système. Il sert de couche entre une application et le système sous-jacent. Par exemple, le système graphique de SDL pourrait utiliser le *frame buffer console* ou X11 sous Linux, mais recourrait à DirectDraw sous Windows. Dans les deux cas, l'API de SDL demeure inchangé, et l'application ne doit pas s'inquiéter de ce qui se passe au-dessous. Dans certains cas, une application SDL soigneusement écrite peut être portée sur une nouvelle plate-forme *via* une simple recompilation rapide.

Dans cet article, nous visiterons entièrement l'API vidéo de SDL. Nous montrerons également comment collecter des entrées à partir du clavier. La majeure partie de cet article a été tirée d'un livre de John Hall, qui sera prochainement disponible sur le développement de jeux Linux (*No Starch Press and Loki Entertainment Software*).

tion l'utilise. Après avoir initialisé l'affichage (c'est-à-dire après avoir créé une fenêtre ou fait basculé la carte vidéo en un mode bien particulier) SDL disparaît de votre chemin, ne fournissant plus qu'un ensemble minimal de fonctions pour diriger les blocs de pixels. SDL n'est pas un outil de dessin, ce que vous faites avec l'appareil vidéo après son initialisation ne regarde pas SDL.

SDL utilise des structures appelées «surfaces» (de caractère **SDL_Surface**) pour représenter les données graphiques. Une surface est simplement un bloc de mémoire où l'on peut enregistrer une région rectangulaire de pixels (points de couleurs individuels). Chaque surface possède une largeur, une hauteur et un format de pixel spécifique (nous en reparlons plus tard). SDL charge des fichiers image directement sur les structures de surface, et l'écran est également une surface (bien que spéciale). La propriété la plus importante des surfaces est qu'elles peuvent être copiées rapidement les unes sur les autres. En fait, les pixels d'une surface peuvent être transférés dans une zone rectangulaire de taille identique d'une autre surface. Cette opération est appelée un «blit» c'est-à-dire un transfert d'images par bloc. Les «blits» sont des éléments fondamentaux de la programmation de jeux parce qu'ils permettent à des images entières d'être composées de graphiques pré-dessinés (souvent créés par des artistes munis de programmes de traitement d'images). Étant donné que l'écran est une surface comme les autres, des images complètes peuvent être envoyées à l'écran par une simple opération de manipulation d'objets graphiques. SDL fournit une fonction générique pour réaliser rapidement des «blits» entre surfaces, et peut même se déplacer au vol entre des surfaces de différents formats de pixels.

Configurer l'affichage

Avant de commencer à manipuler les surfaces sur l'écran, nous devons initialiser la bibliothèque SDL et basculer l'écran dans un mode approprié. Jetez un coup d'œil au **listing 1**, l'équivalent d'un «Hello world !» en SDL.

Configuration de l'affichage

```
#include "SDL.h"
#include <stdio.h>
#include <stdlib.h>
int main()
{
    SDL_Surface *screen;
    /* Initialisation vidéo */
    if (SDL_Init(SDL_INIT_VIDEO) != 0) {
        printf("Unable to initialize SDL: %s\n",
            SDL_GetError());
        return 1;
    };
    atexit(SDL_Quit);
    /* Résolution 640x480 hicolor */
    screen = SDL_SetVideoMode(640,480,16,SDL_FULLSCREEN);
    if (screen == NULL) {
        printf("Unable to set video mode: %s\n",
            SDL_GetError());
        return 1;
    }
};
```

```
};
/* Si tout se déroule normalement : */
printf("Ok\n");
return 0;
}
```

Ce programme renferme le fichier en-tête **SDL.h**, qui est l'en-tête principale de SDL. Chaque application SDL devrait inclure ce fichier. Le programme inclut également deux entêtes standard, pour les fonctions **printf** et **atexit**.

Nous commençons par appeler **SDL_Init** pour initialiser SDL. Cette fonction prend une liste ORed d'arguments pour indiquer quel sous-système devrait être initialisé; nous ne sommes intéressés que par le sous-système vidéo et nous passons donc **SDL_INIT_VIDEO** (si nous avions voulu l'audio, par exemple, nous aurions appelé cette fonction à l'aide de **SDL_INIT_VIDEO | SDL_INIT_AUDIO**). À moins que ne se produise une erreur bloquante, cette fonction devrait retourner un état nul (*via* un **return 0**). Nous utilisons également l'option **atexit** de C pour demander que **SDL_Quit** soit appelé avant que le programme ne se termine. Cette fonction s'assure que SDL peut se fermer correctement (ce qui devient notamment important si une application de plein écran se bloque).

Ensuite, nous utilisons la fonction **SDL_SetVideoMode** pour indiquer l'affichage de la résolution souhaitée (dans ce cas 640 pixels horizontaux sur 480 pixels verticaux) et de la couleur (16 bits). Attention, ce n'est pas si simple que ça : SDL essaiera de configurer l'affichage comme demandé, mais il pourrait échouer. Si cela se produit, SDL ne nous avertira pas mais il émulerà à la place le mode requis en interne. Cela est habituellement acceptable étant donné que le code d'émulation est assez rapide et qu'il vaut mieux ne pas devoir traiter avec des modes multiples en interne. **SDL_SetVideoMode** amène un curseur à la surface qui représente l'écran. Si un problème se passe, la fonction affiche «NULL».

Finalement, en cas de succès nous pouvons sortir. La bibliothèque C appelle automatiquement **SDL_Quit** (puisque nous l'avons enregistré avec **atexit**), et SDL ramène l'affichage vidéo en mode original (Nous pouvons également appeler explicitement **SDL_Quit** si nous voulons fermer le système avant de sortir de l'application; le fait de l'appeler plus d'une fois ne crée aucun problème).

Maintenant que nous avons créé une application SDL, nous devons la compiler. Les applications SDL sont faciles à construire; en présupant que l'installation de SDL soit correcte, elles ne demandent que quelques drapeaux et bibliothèques. La distribution SDL standard comprend un programme appelé **sdl-config** (identique aux programmes **gtk-config** et **glib-config** fournis avec les outils GTK+). Celui-ci fournit les arguments de lignes de commande appropriés à **gcc**. La commande **sdl-config -cflags** produit une liste d'options qui devrait être transmise au compilateur, et **sdl-config -libs** produit une liste de bibliothèques qui devrait y être associée. Nous pouvons mettre cela dans la ligne de commande **gcc**. Si SDL est installé sur votre système, vous pouvez compiler cet exemple avec la commande suivante :

```
$ gcc sdltest.c -o sdltest sdl-config -cflags -libs
```

Dessiner directement des pixels

Mettre des données dans une surface SDL est simple. Chaque structure `SDL_Surface` contient un membre « pixel ». Il s'agit d'un pointeur vide sur une image graphique brute, et nous pouvons écrire directement dessus si nous connaissons le type de pixel pour lequel la surface est configurée. Nous devons appeler la fonction `SDL_locksurface` avant d'accéder à ces données (car certaines surfaces résident dans des zones mémoire spéciales et nécessitent un traitement particulier). Lorsque nous en avons terminé avec la surface, nous devons appeler `SDL_UnlockSurface` pour la libérer. La largeur et la hauteur de l'image sont données par les membres `w` et `h` de la structure, et le format de pixel est spécifié par le membre `format` (qui est de caractère `SDL_PixelFormat`). SDL émule souvent des résolutions d'écran non-standard avec de hautes résolutions, et l'espacement (appelé **pitch member**) de la structure de format de pixel indique la largeur courante du *frame buffer*. Vous devriez toujours utiliser l'espacement (`pitch`) au lieu de `w` pour calculer les déplacements (appelés *offsets*) au sein des tampons de pixels, ou bien votre application pourrait ne pas fonctionner sur certains appareils d'affichage.

L'exemple montré dans le **listing 2** recourra aux informations de format pixel SDL pour dessiner des pixels individuels sur l'écran. Nous avons choisi d'utiliser un mode 16-bits (**hicolor**) pour des besoins de démonstration, mais d'autres modes sont tout aussi simples à programmer.

Dessiner des pixels individuels sur l'écran

```
#include "SDL.h"
#include <stdio.h>
#include <stdlib.h>
Uint16 CreateHicolorPixel(SDL_PixelFormat *fmt,
                          Uint8 red, Uint8
                          green, Uint8 blue)
{
    Uint16 value;
    value = ((red >> fmt->Rloss) << fmt->Rshift) +
            ((green >> fmt->Gloss) << fmt->Gshift) +
            ((blue >> fmt->Bloss) << fmt->Bshift);
    return value;
}
int main()
{
    SDL_Surface *screen;
    Uint16 *raw_pixels;
    int x,y;
    /* "Lock" l'écran */
    SDL_LockSurface(screen);
    /* Recherche d'un pointeur
       sur la mémoire écran. */
    raw_pixels = (Uint16 *)screen->pixels;
    for (x = 0; x < 256; x++) {
        for (y = 0; y < 256; y++) {
            Uint16 pixel_color;
            int offset;
            pixel_color =
                CreateHicolorPixel(screen->format,
```

```
                x,0,y);
            /* Calcul de l'offset */
            offset = (screen->pitch/2 * y + x);
            raw_pixels[offset] = pixel_color;
        }
    }
    /* Déverrouillage de l'écran */
    SDL_UnlockSurface(screen);
    /* Informe SDL qu'une mise à jour.
       Doit avoir lieu */
    SDL_UpdateRect(screen,0,0,0,0);
    /* Pause pour permettre le rafraîchissement */
    SDL_Delay(3000);
    return 0;
}
```

Les commentaires de code donnent les instructions pas à pas, mais certaines choses ne seront peut-être pas évidentes. Le programme emploie une routine très générale pour construire les valeurs de pixels en haute couleur : cette routine fonctionnera avec tout format de pixel de haute couleur (16-bits) reconnu par SDL. Bien que nous aurions pu écrire une routine séparée (plus rapide) pour chaque présentation de données haute couleur, cela demanderait beaucoup de travail et n'améliorerait les performances que de manière marginale. Le format de pixel haute couleur 565 (5 bits rouges, 6 verts et 5 bleus) est peut-être le plus largement utilisé et pourrait raisonnablement être optimisé, mais le 555 et le 555 se retrouvent aussi. De plus, il n'existe aucune garantie que les champs de bit seront dans l'ordre rouge-vert-bleu. Notre routine `CreateHicolorPixel` résout le problème en se référant aux données de la structure `SDL_PixelFormat`. Ainsi, la routine utilise le membre `Rloss` de la structure pour déterminer combien de bits doivent être repris du composant 8-bits rouge et utilise ensuite le membre `Rshift` pour déterminer où les bits rouges devraient être placés dans la valeur pixel 16-bits.

Un autre caractère importante du code source implique la fonction `SDL_UpdateRect`. Comme nous l'avons mentionné plus tôt, SDL émule parfois les modes vidéo si la carte vidéo est incapable de fournir elle-même un certain mode. Par exemple, si la carte vidéo ne supporte pas un mode 24-bits requis, SDL pourrait sélectionner un mode 16-bits à la place et retourner une configuration de *frame buffer* simulée pour des pixels 24-bits. Ceci permettrait à votre programme de poursuivre normalement, et SDL gèrerait au vol la conversion de 24-bits en 16-bits (avec une légère perte de performance). La fonction `SDL_UpdateRect` informe SDL qu'une portion de l'écran a été remise à jour et qu'elle devrait réaliser les conversions appropriées pour afficher cette zone. Si un programme n'utilise pas cette fonction, il se pourrait que ça continue à fonctionner. Il vaut mieux mettre toutes les chances de son côté et appeler cette fonction lorsque la surface du *frame buffer* a été modifiée.

Finalement, si vous exécutez le programme, vous pourriez remarquer qu'il s'exécute dans une fenêtre au lieu de prendre l'intégralité de l'écran. Pour changer cela, remplacez le zéro de l'appel `SDL_SetVideoMode` avec la constante `SDL_FULLSCREEN`. Cependant, soyez prudent ; les applica-

tions en plein écran sont difficiles à « déboguer ». De plus, elles ont tendance à compliquer beaucoup les choses lorsqu'elles se bloquent.

Dessiner avec des « blits »

Nous avons vu auparavant comment dessiner directement des pixels sur une surface, et il n'y a aucune raison que l'on ne puisse créer un jeu entier rien qu'avec cela. Cependant, il existe un meilleur moyen de dessiner de grandes quantités de données sur l'écran. Notre prochain exemple chargera une surface entière à partir d'un fichier et la dessinera à l'aide d'une simple fonction de copiage de surface SDL. Le code apparaît dans le **listing 3**.

Dessiner de grandes quantités de données sur l'écran

```
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    SDL_Surface *screen;
    SDL_Surface *image;
    SDL_Rect src,dest;
    /* Initialisation comme vu précédemment */
    /* Charge le fichier bitmap. SDL_LoadBMP retourne
       un nouveau pointeur mémoire sur ces données */
    image = SDL_LoadBMP("tux.bmp");
    if (image == NULL) {
        printf("Unable to load bitmap.\n");
        return 1;
    };
    src.x = 0; src.y = 0;
    src.w = image->w; /* copie l'image entière */
    src.h = image->h;
    dest.x = 0; dest.y = 0;
    dest.w = image->w;
    dest.h = image->h;
    /* La fonction SDL_BlitSurface va
       dessiner l'image à l'écran */
    SDL_BlitSurface(image,&src,screen,&dest);
    /* Mise à jour de l'écran. */
    SDL_UpdateRect(screen,0,0,0,0);
    /* Pause */
    SDL_Delay(3000);
    /* Libère l'espace mémoire */
    SDL_FreeSurface(image);
    return 0;
}
```



Comme vous pouvez le constater, le fichier bitmap est chargé en mémoire avec la fonction **SDL_LoadBMP**. Cette fonction retourne un pointeur sur une structure **SDL_Surface** contenant l'image, ou un pointeur **NULL** si l'image ne peut être chargée. Lorsque le fichier a été correctement chargé, le bitmap est représenté comme une surface SDL ordinaire, et un programme peut le dessiner sur l'écran ou tout autre surface. Les bitmaps utilisent une mémoire allouée dynamiquement, et devraient être libérés lorsque l'on n'en a plus besoin. La fonction **SDL_FreeSurface** libère la mémoire allouée à un bitmap.

La fonction **SDL_BlitSurface** réalise un *blit* d'une surface sur une autre, passant d'un format de pixel à un autre si nécessaire. Cette fonction prend quatre arguments : une surface source (l'image à partir de laquelle on doit copier), une structure **SDL_Rect** définissant la région rectangulaire de la surface source à copier, une surface de destination (l'image sur laquelle on doit copier), et une autre structure **SDL_Rect** indiquant les coordonnées de la destination à laquelle l'image devrait être dessinée. Ces deux rectangles doivent être de la même largeur et de la même hauteur (SDL ne réalise pas pour l'instant d'étirement), mais les coordonnées de départ x et y des régions peuvent être différentes.

Clés de couleur et transparence

Les jeux doivent souvent simuler la transparence. Par exemple, supposons que nous avons un bitmap d'un personnage de jeu contre un arrière-plan solide, et que nous voulons dessiner le personnage dans un niveau de jeu. Cela semblerait stupide de dessiner le personnage tel quel ; l'arrière-plan serait également dessiné et le personnage serait entouré d'un bloc de couleur solide. Il vaudrait mieux dessiner seulement les pixels qui font partie du personnage et non pas ceux de l'arrière-plan solide. Nous pouvons réaliser cela à l'aide d'un *blit* de clé de couleur. SDL fournit un support pour cela et même un support pour une compression de type **run-length** (un chouette truc pour accélérer le dessin). L'accélération RLE offre une importante augmentation de performances pour la manipulation d'images réalisées par clés de couleur, mais ce n'est pratique que dans les cas de bitmaps qui ne seront pas modifiés durant l'exécution du programme (puisque la modification d'image RLE nécessite de décompresser et recompresser l'image).

Une clé de couleur est une valeur de pixel particulière qu'un programme déclare être transparente (en SDL, cette opération est réalisée par la fonction **SDL_SetColorKey**).

Dans notre exemple de personnage d'un jeu, nous avons placé la clé de couleur sur la couleur de l'arrière-plan solide et elle n'avait pas été dessinée. Les clés de couleur facilitent la combinaison d'images rectangulaires d'objets non-rectangulaires.

Dans le prochain exemple, nous utiliserons un *blit* de clés de couleur pour dessiner l'image d'un Tux contre une autre image (voir le **listing 4**). Tux est mis en mémoire contre un arrière-plan solide bleu et nous utiliserons donc le bleu (RGB 0, 0, 255) en tant que clé de couleur. En comparaison, nous dessinerons aussi la même image de pingouin sans clé de couleur.

Utilisation de clés de couleur

```
#include <SDL/SDL.h>
#include <stdio.h>
#include <stdlib.h>

int main()
```

```

{
  SDL_Surface *screen;
  SDL_Surface *background;
  SDL_Surface *image;
  SDL_Rect src,dest;
  Uint32 colorkey;

  /* NDT: ATTENTION Vous devez initialiser ici
  un écran en 640x480 */

  /* Charge le fichier bitmap. */
  background = SDL_LoadBMP("background.bmp");
  if (background == NULL) {
    printf("Unable to load bitmap.\n");
    return 1;
  };

  image = SDL_LoadBMP("tux.bmp");
  if (image == NULL) {
    printf("Unable to load bitmap.\n");
    return 1;
  };

  /* Dessine un fond d'écran. */
  src.x = 0; src.y = 0;
  src.w = background->w;
  src.h = background->h;
  dest.x = 0; dest.y = 0;
  dest.w = background->w;
  dest.h = background->h;
  SDL_BlitSurface(background,&src,screen,&dest);

  /* Dessine le Tux sans l'utilisation d'une clé de couleur. */
  src.x = 0;
  src.y = 0;
  src.w = image->w;
  src.h = image->h;
  dest.x = 30;
  dest.y = 90;
  dest.w = image->w;
  dest.h = image->h;
  SDL_BlitSurface(image,&src,screen,&dest);

  /* Idem mais avec une clé de couleur */
  colorkey = SDL_MapRGB(image->format,0,0,255);

  SDL_SetColorKey(image,SDL_SRCCOLORKEY,colorkey);
  src.x = 0; src.y = 0;
  src.w = image->w;
  src.h = image->h;
  dest.x = screen->w - image->w - 30;
  dest.y = 90;
  dest.w = image->w; dest.h = image->h;
  SDL_BlitSurface(image,&src,screen,&dest);

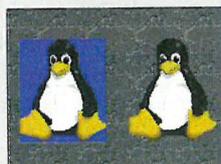
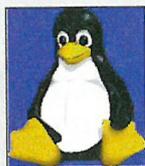
  /* Mise à jour de l'écran. */
  SDL_UpdateRect(screen,0,0,0,0);

  /* Pause. */
  SDL_Delay(10000);

  /* Libère les ressources. */
  SDL_FreeSurface(background);
  SDL_FreeSurface(image);

  return 0;
}

```



Manipulation de clavier simple

SDL assigne un code «*virtual keysym*» à chaque touche du clavier. Ces codes (qui sont des nombres entiers) établissent une correspondance à un certain niveau avec les codes **scan** du clavier du système d'exploitation (en fait qui établit une correspondance avec les codes produits par le matériel clavier), mais SDL s'occupe de cette mise en correspondance de manière transparente. SDL fournit un symbole de pré processeur pour chaque *keysym* virtuelle. Ainsi, la touche «Escape» correspond au symbole **SDLK_ESCAPE** (vous pouvez découvrir une liste complète de *keysyms* en parcourant la documentation de SDL). Nous utilisons ces codes à chaque fois que nous avons besoin de contrôler directement l'état d'une touche (enfoncée ou pas). Les codes virtuels *keysyms* sont représentés par les caractères de données **SDLKey**.

Étant donné que nous ne nous occuperons pas aujourd'hui de l'interface d'événement (en fait, nous ne l'avons pas encore réellement mentionnée), nous devons demander explicitement au clavier son état courant lorsque nous aurons besoin d'informations sur une touche. Un programme peut obtenir un *snapshot* d'un clavier complet sous la forme d'un tableau. La fonction **SDL_GetKeyState** retourne un pointeur vers le tableau interne d'état du clavier de SDL, qui est répertorié avec les constantes **SDLK_Keysym**. Vous devez simplement appeler cette fonction une seule fois, le pointeur reste valide pour toute la durée du programme. Chaque entrée dans le tableau est un simple drapeau **Unit8** indiquant si cette touche est hors fonction pour l'instant. Vous devriez appeler périodiquement **SDL_PumpEvents** pour mettre à jour les données du tableau.

Encore plus !

Cela devrait être suffisant pour que vous démarriez avec SDL. Nous avons laissé tombé pas mal de choses, notamment l'animation, l'*alpha blending* et le *playback* audio. Si vous souhaitez en savoir plus sur la programmation à l'aide de cette bibliothèque, le meilleur endroit pour débiter vos recherches est le *SDL Documentation Project* sur <http://libsdl.org/>. Vous pouvez également vous arrêter sur le **#sdl channel** sur irc.openprojects.net, où vous trouverez probablement un bon nombre de fans de SDL ayant pas mal d'expérience. Amusez-vous bien et bon *coding*! ■



L'auteur :

John Hall est un étudiant spécialisé dans les sciences informatiques de l'Université de Georgia Tech et est très intéressé par les jeux Linux. Lorsqu'il n'est pas en transe devant un clavier d'ordinateur, on peut souvent le trouver en train de jouer à l'épée dans le campus ou de prendre soin de ses araignées domestiques. On peut le contacter à overcode@lokigames.com.

Ruby can't fail, il fait tout!

Prochain grand rival de Perl, Ruby, est le petit japonais qui monte. Ce langage distribué sous licence GPL gère également les scripts.

Maurice Szmurlo
Maurice.Szmurlo@info.unicaen.fr



Sur votre CD-Rom



La page d'accueil de Ruby est située à l'URL suivante : <http://www.ruby-lang.org/>. On y trouvera les distributions en sources ou exécutables pour différents systèmes, les documentations et des liens vers un répertoire d'applications et de bibliothèques. Les documentations sont pour l'instant assez spartiates : la documentation officielle s'applique à la version 1.4 alors que son niveau de version actuel est 1.6.1. Par contre, celle des bibliothèques standard est très complète. Vous trouverez une *mailing list* à l'adresse suivante : ruby-talk-ctl@netlab.co.jp. Les problèmes de syntaxe ou les *bugs* restent rarement sans solution. Notons finalement l'existence de deux livres en anglais sur Ruby : *Programming Ruby: A Pragmatic Programmers Guide*, David Thomas et Andrew Hunt, Addison-Wesley Pub Co ; *The Ruby Programming Language*, Yukihiro «Matz» Matsumoto, Addison-Wesley Pub Co (à paraître ce début d'année). Cet article est le premier d'une série dont l'objectif est de présenter les différents aspects de ce langage : la syntaxe et les structures de données de base, la programmation objet, la programmation multi-tâches, les bibliothèques annexes, etc. Commençons par une présentation très générale pour une prise en main rapide.

Ruby est un langage interprété qui incorpore des caractéristiques de haut niveau empruntées à tout un ensemble de langages d'intelligence artificielle (Scheme, CLU,...), mais également aux langages de script tels que Perl. Il est orienté objet, utilise les *threads* de manière indépendante du système d'exploitation employé, gère les exceptions, intègre la notion *et*, et est extensible au *runtime* tant au niveau des classes que des objets pris individuellement. Sa syntaxe très claire est inspirée d'Eiffel. Il est portable (Unix, Windows, BeOS, Mac) et facilement extensible par ajout de bibliothèques dynamiques.

Ruby est également un langage de script disposant par défaut de nombreuses bibliothèques : gestion des chaînes de caractères, expressions régulières, accès réseau,... Enfin, Ruby est gratuit : il est distribué sous licence GPL. Ruby est un langage qui «monte» et suscite l'enthousiasme de toute une communauté. Au Japon, son pays de naissance, il fait concurrence à Perl à tel point qu'il fera l'objet de présentations spéciales à la prochaine conférence sur Perl.

Installation, création et exécution de programmes

L'installation de Ruby sous Windows est très simple : elle s'effectue par **Install Shield**. Sous Unix, il suffit de télécharger et de décompresser l'archive contenant le code source, puis taper sous *shell* : **./configure; make; make install**. Le fichier **README** dans le répertoire de décompression décrit ce processus en détail.

Une fois installé, Ruby peut être appelé sur la ligne de commande. **ruby -e <expression>** exécute l'expression donnée

en paramètre. Les programmes plus longs seront sauvegardés dans un fichier et seront lancés par la commande `ruby <nom_de_fichier>`. Si le nom de fichier n'est pas donné sur la ligne de commande, Ruby lit le source sur son entrée standard. Pour une utilisation interactive, la distribution fournit le programme `irb.rb` situé dans le répertoire des bibliothèques de Ruby.

Variables et fonctions

En Ruby, il n'est pas nécessaire de déclarer les variables utilisées. Une variable est définie lors de sa première affectation. Le type d'une variable est déterminé en fonction du type de la donnée qui est affectée. `a="Bonjour"` initialise la variable `a` à la chaîne de caractères « Bonjour » ; `a=10` réinitialise cette même variable à l'entier 10. En Ruby, toute donnée manipulée est un objet. À tout moment, il est possible de connaître le type d'un objet en y appliquant la méthode `type`. Ainsi, `puts a.type` affichera `Fixnum` qui est l'un des types d'entiers connus de Ruby. Notons que les affectations multiples sont autorisées : `a, b = 1, "toto"`, affecte `1` à la variable `a` et la chaîne « toto » à `b`.

Un nom de variable peut être préfixé par les caractères `$` ou `@`. `$var` est une variable globale, `@var` est un attribut de classe. Une variable non-préfixée a une portée locale ou bien bloque où elle a été utilisée pour la première fois.

Une fonction en Ruby est introduite par le mot clef `def` et sa définition se termine par `end`. `def` est suivi du nom de la fonction et, éventuellement, de la liste des paramètres formels qui sont donnés entre parenthèses ou pas. L'exemple suivant définit la fonction `cube` qui renvoie le cube de son paramètre. Pour afficher le cube de 3, il suffit d'écrire :

```
puts cube(3).
def cube(x)
  a = x * x * x
  return (a)
end
```

Une fonction peut également renvoyer plusieurs valeurs :

```
def carre_et_cube(x)
  return (x*x, x*x*x)
end
a, b = carre_et_cube(2)
puts a, b
affichera 4 et 8.
```

Comme la plupart des langages de programmation modernes, Ruby est récursif. Attention : la récursivité est un processus très coûteux en terme de CPU. La fonction suivante renvoie la valeur de `x` puissance `y`.

```
def x_puiss_y (x, y)
  if y == 1 then
    return x
  end
  return x * x_puiss_y(x, y-1)
end
```

Les concepteurs de Ruby ont voulu un langage très souple au niveau de la syntaxe afin que les programmeurs venant de mondes différents puissent facilement prendre leurs marques. Par exemple, les parenthèses autour des paramètres formels d'une fonction ne sont pas nécessaires. Il en va de même pour le `then` du branchement conditionnel. Enfin, toute fonction renvoie le résultat de sa dernière évaluation. Ainsi le corps de la fonction `cube`, `return (x*x*x)`, pourrait être remplacé par `x*x*x`. Certaines formes syntaxiques sont cependant plus pénalisantes en terme de vitesse que d'autres. Par exemple, la forme suivante de l'exponentiation récursive est environ 25 % plus rapide que celle présentée plus haut :

```
def x_puiss_y_bis (x, y)
  if y==1
    x
  else
    x * x_puiss_y_bis(x, y-1)
  end
end
```

Au fur et à mesure de l'apprentissage du langage il est donc nécessaire de s'adapter de plus en plus à la syntaxe et aux structures de contrôle les plus performantes. Les débutants choisiront plutôt les formes qui leur conviennent le mieux.

Types de base

Types numériques

Ruby manipule deux types numériques : les entiers et les nombres flottants.

Il y a deux catégories d'entiers : les *Fixnums* et les *Bignums*. Les *Fixnums* sont des entiers représentés par des « mots machine ». Les valeurs des *Fixnums* se situent dans l'ensemble $\{-2^{30}, \dots, 2^{30}-1\}$. Les *Bignums* sont, quant à eux, des entiers de taille quelconque, dans la limite de la mémoire disponible, bien entendu. Les conversions entre les *Fixnums* et les *Bignums* sont transparentes : si le résultat d'une opération entre deux *Fixnums* sort de l'ensemble de définition, il est converti en *Bignum*. Inversement, si le résultat d'une opération utilisant les *Bignums* est dans l'ensemble de définition des *Fixnums*, il est converti en *Fixnum*. La capacité à gérer de très grands nombres est extrêmement pratique, mais se paie au prix de la performance.

Les opérations sur les nombres entiers sont :

- Les opérations arithmétiques (+, -, *, /, modulo : %, exponentiation : **).
- Les opérations sur les bits (inversion : ~, ou : |, et : &, xor : ^, décalage à gauche <<, décalage à droite : >> et l'accès au i-ème bit : [i]).

Les opérateurs arithmétiques et les opérations sur les bits combinés avec l'affectation, `var1 op= var2` sont un raccourci de `var1 = var1 op var2`. Par exemple, `a+=2` ajoute 2 à la variable `a`.

- Les comparaisons : !=, <, <=, ==, >=, >, <=>. Ce dernier opérateur renvoie -1, 0, ou 1 selon que l'opérande de gauche est plus petit, égal ou plus grand que l'opérande de droite.

Des ensembles de nombres entiers consécutifs peuvent être définis par des objets de type *Range* : `i1..i2` représente l'ensemble des entiers entre `i1` et `i2`. Les *Range* sont très utiles pour spécifier les bornes des boucles ou un ensemble d'indices d'un tableau ou d'une chaîne de caractères.

Les nombres flottants, les *Float*, ont une représentation native (comme les *Fixnums*) dépendante du système. Les opérations sur les *Float* sont les mêmes que celles sur les entiers plus quelques fonctions spécifiques comme par exemple les arrondis (*ceil*, *round*, *floor*), les tests d'intégrité (la valeur est-elle finie : *finite?* ou infinie : *infinite?*).

Le code ci-dessous illustre certaines manipulations décrites dans cette section :

```

irb(main):001:0> a = 2**30
irb(main):002:0> puts a.type
Bignum
irb(main):003:0> a -= 1
irb(main):004:0> puts a.type
Fixnum
irb(main):005:0> a = a / 5.5
irb(main):006:0> puts a.type
Float
irb(main):007:0> puts a.finite?
true
    
```

Pour faciliter la lecture des exemples, des lignes affichant les évaluations inutiles ont été supprimées. De plus, dans ce qui suit, le long *prompt* d'irb (`irb(main):001:0>`) sera remplacé par `ruby>`.

Les booléens

Comme la plupart des langages, Ruby définit deux valeurs pour les booléens : *true* et *false*. Chacun de ces deux objets est l'unique instance des classes *TrueClass* et *FalseClass*, respectivement. *True* et *false* sont renvoyés par les prédicats et les comparaisons et sont utilisés dans les conditions. Les opérations logiques sont le «et» (`&&`) et le «ou» (`||`). Une facilité apportée par Ruby, et chère aux lispiciens, est la valeur *nil* (unique instance de la classe *NilClass*). *nil* représente «quelque chose qui n'existe pas», par exemple, la valeur d'une variable référencée mais à laquelle aucune valeur n'a été donnée explicitement. Ce cas de figure peut survenir lors d'une affectation multiple : `a, b, c = 1, 2`, la variable `c` est bien référencée, mais aucune valeur ne lui est affectée : le système lui affecte alors par défaut la valeur *nil*. `puts a, b, c` donnera `1, 2` et *nil*. Par défaut, *nil* est faux et toute autre valeur que *nil* est vraie (sauf *false*, bien entendu). Attention : bien que *nil* et *false* représentent «faux», il s'agit d'objets différents appartenant à des classes différentes, la valeur de *nil* est différente de la valeur de *false*, ou en d'autres termes `nil == false` s'évalue à *false*. Ceci peut conduire à des incohérences (*a priori*) du style :

```

a = 1.5
if a.infinite? == false
  puts "a est fini"
else
  puts "a est infini"
end
    
```

qui affiche que `a` est infini ! Il est conseillé d'écrire des tests sous la forme `if expression ... end` plutôt que `if expression == true ... end` ou `if expression == false ... end`.

Les chaînes de caractères (classe String)

L'une des forces de Ruby est sa capacité à manipuler les chaînes de caractères. Une chaîne de caractères est un ensemble de caractères compris soit entre guillemets soit entre apostrophes. «*bonjour*» ou '*bonjour*' sont deux chaînes. La différence est qu'il est possible d'inclure dans une chaîne définie entre guillemets des caractères d'échappement ou des résultats d'évaluation. Les caractères d'échappement sont introduits par un antislash (`\`), comme par exemple le traditionnel fin de ligne `\n` ; les évaluations d'expressions sont introduites par `#{...}`. Quelques exemples sont donnés ci-dessous :

```

ruby> puts "a\nb\nc"
a
b
c
ruby> puts 'a\nb\nc'
a\nb\nc
ruby> puts "resultat: 3+3=#{3+3}"
resultat: 3+3=6
ruby> puts 'resultat: 3+3=#{3+3}'
resultat: 3+3=#{3+3}
    
```

Deux *strings*, `s1` et `s2`, peuvent être comparés par `!=`, `<`, `<=`, `==`, `=>`, `<=>`. Pour la concaténation, il existe deux opérateurs : `+` et `<<`. `s1+s2` renvoie un nouvel objet, alors que `s1<<s2` concatène `s2` directement à `s1`. Une opération particulièrement utile est la concaténation multiple : si `i` est un entier, `s1*i` renvoie une chaîne composée de `i` occurrences de `s1`.

Par exemple `"b"*3` produit `"b b b"`. L'extraction de sous-chaînes est réalisée par l'opérateur `[]`. Les caractères des chaînes commencent à l'indice zéro. `s1[i]` renvoie par conséquent le `(i+1)`-ème caractère de la chaîne. `"abc"[0]` renvoie «`a`», le premier ; `"abc"[1]` renvoie «`b`», le deuxième ; `"abc"[4]` renvoie *nil*. Le paramètre de `[]` peut également être un *Range*, `i1..i2`. Dans ce cas, c'est la sous-chaîne commençant en `i1` et se terminant en `i2` qui est renvoyée : `"abcdf"[2..4]` renvoie `"ced"`. Le remplacement est réalisé par `[]=`. `s1[i1]=s2` remplace le `(i1+1)`-ème caractère de `s1` par la chaîne `s2`. Dans la même logique, `s1[i1..i2]=s2` remplace la sous-chaîne d'indices `i1` à `i2` par `s2`.

Outre les opérateurs, la classe *String* fournit un grand nombre de méthodes utilitaires de recherche de sous-chaînes, de formatage ou de recherche et remplacement. La plupart peuvent être réécrites en employant les opérateurs donnés ci-dessus. Elles sont listées dans la documentation de la bibliothèque standard. Ces méthodes viennent en général sous deux formes : `s1.methode` ou `s1.methode!`. La première forme produit une nouvelle chaîne alors que la deuxième modifie directement `s1`. Cette manière de nommer les méthodes n'est pas particulière aux *strings* : de façon générale, toute méthode modifiant l'objet sur lequel elle s'applique se termine par un «`!`», de même que les prédicats se terminent par un «`?`».

Les expressions régulières (classe Regexp)

Les expressions régulières (on parle également de *pattern*) sont définies entre deux caractères `/` `/d/` *match* toutes les chaînes de caractères contenant au moins un caractère «`d`». L'opérateur de *matching* de base en Ruby est `=~`, et sa négation est `!~`. Si `s` est une *string* et `r` une *Regexp*, `s =~ r` (ou `r =~ s`) renvoie la première position dans `s` à partir de laquelle le *pattern* `r` a été repéré. Si `r` n'est pas trouvé, `nil` est renvoyé. Voici quelques exemples d'expressions régulières :

1. chaîne contenant un `e`, suivi d'un nombre quelconque de caractères, suivis d'un autre `e`; ce *pattern* pouvant être placé à n'importe quelle position dans la chaîne : `/e.*e/`;
2. le *pattern* défini en 1. doit commencer la chaîne : `/^e.*e/`; doit terminer la chaîne : `/e.*e$/`; représente la chaîne entière : `/^e.*e$/`;
3. la chaîne est un nombre non-signé positif : `/^[0-9]+$/`; contient un tel nombre : `/+[0-9]+ +/` (un espace ou plus en début et en fin du nombre) ;
4. la chaîne contient un nombre hexadécimal : `/0(x|X)([0-9]|[a-f]|[A-F])+/`.

Le résultat de la recherche d'une *Regexp* dans une chaîne par la méthode `match` (`r.match(s)`) est mémorisé dans un objet de type `MatchData`. Ces objets sont extrêmement pratiques lors de l'analyse de chaînes car ils contiennent les indices de début et de fin des apparitions de toutes les parties de la *Regexp* (méthodes `begin` et `end`).

Les tableaux et tables de hashage

Un tableau (classe `Array`) est une collection ordonnée par des entiers (*Fixnum*) d'objets. Le premier élément d'un tableau est à l'indice 0. Les tableaux peuvent être créés de manière déclarative ou impérative. La syntaxe déclarative liste les éléments initiaux du tableau, par exemple : `tab = [1, "a", [1, 2], 1.3]`. Le premier élément de `tab` est un entier (indice 0), le second est une chaîne, le troisième un tableau, etc. Dans la syntaxe impérative, on écrira : `tab = Array.new`. Notons que cette syntaxe, `NomClasse.new`, est générique à (pratiquement) toutes les classes prédéfinies (essayez `s = String.new("abc")` ou `r = Regexp.new(/^a$/)`) ou définies par l'utilisateur. Quand aucun argument n'est passé à `Array.new`, un tableau vide est fabriqué (équivalent à `[]`). `Array.new(3)` fabrique un tableau de 3 éléments tous positionnés à `nil` (i.e. : `[nil, nil, nil]`); `Array.new(5, "A")` fabrique `["A", "A", "A", "A", "A"]`. Comme pour les chaînes de caractères, l'accès en lecture aux éléments d'un tableau se fait par l'opérateur `[]` et l'accès en écriture par `[]=`. `tab[1]` est le deuxième élément du tableau. `tab[0..3]` renvoie un tableau contenant les quatre premiers éléments de `tab`. `tab[0]="a"` remplace le premier élément de `tab` par la chaîne «`a`», et `tab[0..2]="a"` remplace les trois premiers éléments de `tab` par «`a`». Si l'indice (ou l'intervalle) donné à `[]=` dépasse la taille du tableau, des éléments positionnés à `nil` sont insérés automatiquement.

```
ruby> tab = [1, 2, 3, 4, 5]
ruby> p tab
[1, 2, 3, 4, 5]
ruby> tab[0] = "a"
```

```
ruby> p tab
["a", 2, 3, 4, 5]
ruby> tab[0..1] = ["A", "B", "C"]
ruby> p tab
["A", "B", "C", 3, 4, 5]
ruby> tab[10] = "Z"
ruby> p tab
["A", "B", "C", 3, 4, 5, nil, nil, nil, nil, "Z"]
```

Contrairement au `puts` qui affiche un objet sous forme de chaîne de caractères, la commande `p` affiche un objet sous sa forme canonique.

Les opérations sur les tableaux sont très nombreuses. `&` effectue une intersection entre deux tableaux, `|` effectue l'union, `-` effectue la différence. Il s'agit là d'opérations ensemblistes, c'est-à-dire que les doublons sont effacés. `+`

concatène deux tableaux, `<<` ajoute un élément à la fin, et `==`, `!=`, et `<=>` permettent d'effectuer des opérations de comparaison. Certaines méthodes permettent d'utiliser les tableaux comme des piles (`push`, `pop`) ou comme des listes de propriétés à la Lisp (`assoc`, `rassoc`). Les tableaux peuvent être triés (méthode `sort`) si les éléments qu'ils contiennent sont comparables par l'opérateur `<=>`.

Les cousins des tableaux sont les tables de «hashage» ou tableaux associatifs (classe `Hash`). Un tableau associatif est un tableau dont les indices, (ou les clefs), ne sont pas des entiers mais des objets quelconques. La syntaxe déclarative d'un tel tableau utilise des accolades plutôt que des crochets. Ainsi, `htab = {}` produit un tableau vide et `htab = {"a"=>"lettre 'a'", "b"=>"lettre 'b'"}` produit un tableau de deux éléments dont les clefs sont les chaînes «`a`» et «`b`» et les valeurs respectives sont «`lettre 'a'`» et «`lettre 'b'`».

Les structures de contrôle

Ruby propose une large panoplie de structures de contrôle ; exécutions conditionnelles, boucles, itérateurs et exceptions.

Exécution conditionnelle : if et case

La syntaxe du `if` est la suivante : `if <cond> [then] <corps_1> [else <corps_2>]`. Le `then` est optionnel. Si plusieurs conditions exclusives doivent être prises en compte, on peut utiliser `elsif` : `if <cond_1> <corps_1> elsif <cond_2> <corps_2> ... [else <corps_sinon>]`.

Le `case` permet d'effectuer des choix multiples de manière très souple. La syntaxe la plus simple est la suivante :

```
case <expr>
when <expr_1> [then]
  <corps_1>
...
when <expr_i> [then]
  <corps_i>
...
[ else
  <corps_sinon> ]
end
```

expr est une variable ou une expression. Sa valeur est comparée en séquence aux **expr_i**. Le corps **i**, correspondant au premier **expr_i** pour lequel la comparaison est vraie, est exécuté et le contrôle sort du **case**. Si aucune des comparaisons n'est vérifiée, le contrôle exécute **corps_sinon** si le **else** est spécifié.

Plusieurs expressions peuvent suivre chaque **when** : le **corps** correspondant est exécuté si l'une de ces expressions est vérifiée (voir l'exemple ci-dessous).

Une caractéristique intéressante du **case** est la méthode utilisée pour effectuer la comparaison : l'opérateur **===**, dont nous n'avons pas encore parlé. Pour la plupart des classes présentées dans les sections précédentes, l'opérateur **===** est équivalent à **==**. Lors de la définition d'une nouvelle classe, il est possible de redéfinir cet opérateur afin de lui donner une sémantique particulière. Par exemple, pour les *Range* et les entiers, l'opérateur **===** teste si l'entier est élément de l'intervalle. Ainsi **(1..3) === 2** s'évalue à **true** alors que **(1..3) === 4** s'évalue à **false**. Pour les *Regex*, **===** est équivalent de **=~**.

```
def tester_valeur (a)
  case a
  when 2
    puts "a == 2"
  when 1, 3..4
    puts "a == 1 ou a entre 3 et 4"
  when 5, 10, 15
    puts "a == 5 ou a == 10 ou a == 15"
  else
    puts "autre valeur"
  end
end
```

Les boucles while et for

La boucle **while** a la syntaxe suivante: **while <cond>** **<corps>** **end**. Bien entendu si **corps** ne modifie pas les variables utilisées dans la condition, la boucle est infinie.

La boucle **for** permet de parcourir une collection. Sa syntaxe est la suivante : **for o in collection corps end**.

Nous avons décrit un certain nombre de collections dans les sections précédentes : le *Range* est une collection d'entiers et les *Array* ou les *Hash* sont des collections d'objets quelconques.

Le **for** permet d'accéder à tous les éléments d'une collection quelconque de manière générique. La fonction suivante en est un exemple :

```
def afficher_collection (collection)
  i = 0
  print "Collection de type #{collection.type} -> "
  p collection
  for o in collection
    print "Element #{i} de type #{o.type} -> "
    p o
    i += 1
  end
end
afficher_collection (1..10)
afficher_collection ["a", 12, 1.9, ["a", "b", "c"]]
afficher_collection ({"1er"=>"a", "2e"=>12, "3e"=>1.9, "4e"=>["a", "b", "c"]})
```

Cette manière, beaucoup plus générale, de considérer la boucle **for** présente dans de nombreux langages de haut niveau est appelée itérateur.

Ruby dispose d'une structure de boucle infinie **loop { corps }** et qui est équivalente à **while true corps end**. Notez cependant la différence de syntaxe du *loop* qui utilise des accolades (voir plus bas).

Toute boucle peut être interrompue de quatre manières :

- **break** a pour effet de sortir de la boucle courante;
- **next**, l'équivalent du « continue » du C/C++/Java, fait sauter le contrôle à la fin de l'itération courante;
- **redo**, recommence l'itération courante;
- **return** fait sortir le contrôle non seulement de la boucle courante, mais également de la fonction qui la contient et renvoie une valeur.

Les « modifieurs »

Pour alléger dans des cas simples une syntaxe qui requiert des blocs, le **if** et le **while** peuvent être utilisés sous forme de *modifieurs* d'une commande. Par exemple, au lieu d'écrire :

```
if debug
  trace(...)
end
```

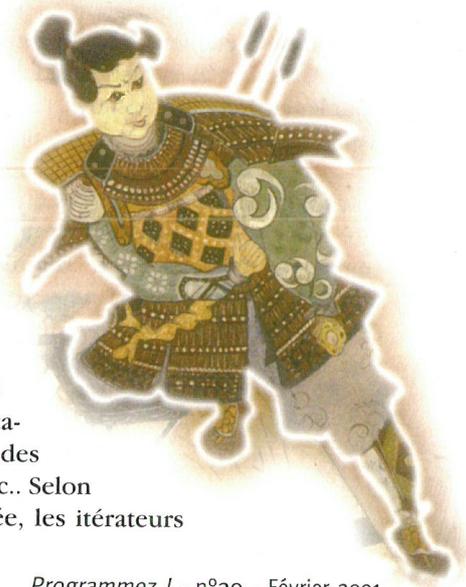
on peut écrire: **trace(...) if debug**. De manière plus générale, la syntaxe est la suivante: **expr if cond**. L'expression **expr** n'est exécutée que si la condition **cond** est évaluée à **true**. De même, une répétition simple peut être codée par: **expr while cond**. Par exemple, si **i** vaut 0, **puts i += 1 while i < 10** affiche les entiers de 1 à 10.

Les itérateurs

Un itérateur est une structure permettant de parcourir une collection, élément par élément, et de leur appliquer un traitement. Les tableaux (de même que virtuellement tous les types en Ruby) définissent la méthode **each** qui permet de réaliser le parcours en utilisant la syntaxe suivante:

a.each { |e| <travailler sur e> }. De manière plus formelle, **each** est suivi d'un bloc d'instructions entre accolades qui

sera exécuté pour chaque élément. À l'intérieur de ce bloc d'instructions, l'élément courant est accessible par une variable dont le nom est spécifié entre deux caractères | (pipe) en tête de bloc. **each** s'applique aux *Range*: **(1..10).each { |i| puts i }**, aux *Hash*: **{"a"=>1, "b"=>2}.each { |c| p c }** (**c** a pour valeurs des tableaux composés des couples [clef, valeur]), etc.. Selon le type de donnée traitée, les itérateurs



peuvent prendre des formes différentes. Par exemple, pour les *Hash*, il existe l'itérateur `each_pair { |c, v| ... }` qui pour chaque élément affecte la valeur de la clef à `c` et la valeur correspondante à `v`. Si `tab` est un tableau, `tab.collect{...}` fabrique un nouveau tableau en se basant sur les éléments de `tab`, `tab.collect!{...}` modifie directement les éléments de `tab`. D'autres itérateurs sont décrits dans la documentation.

Ruby offre au développeur la possibilité de définir ses propres itérateurs *via* le mot-clef `yield`. Si `yield` est présent dans la définition d'une fonction, son exécution consiste à donner le contrôle au bloc d'instructions entre accolades qui suit l'appel de la fonction. Si des paramètres sont fournis à `yield`, il seront passés au bloc. `yield` permet cependant beaucoup plus. Définissons, par exemple, un nouveau type de boucle, `repeat`, qui exécute son `corps` un nombre donné de fois: `repeat(3) { puts "toto_#{i}" }` affichera: «`toto_1`», «`toto_2`», «`toto_3`». La définition de `repeat` est la suivante:

```
def repeat (n)
  for i in 1..n
    yield (i)
  end
end
```

Les exceptions

Ruby gère les éventuelles erreurs qui peuvent survenir dans un programme par le biais des exceptions. En Ruby, une exception est un objet (classe `Exception`) qui véhicule les informations sur le fichier, la fonction, le numéro de ligne où est survenue l'erreur ainsi qu'une chaîne de caractères décrivant celle-ci.

Une exception est lancée par `raise` et est «attrapée» dans un bloc `begin ... rescue ... ensure ... end`, qui est sensiblement équivalent au `try{ ... } catch(...) { ... } finally { ... }` de Java. Pour donner une idée de leur fonctionnement, essayez la fonction suivante:

```
def exceptionFunc (v)
  case (v)
  when 1
    puts "on lance exception_1"
    raise "exception_1"
  when 2
    puts "on lance exception_2"
    raise "exception_2"
  end
  puts "fonctionnement normal -> v=#{v}"
end
```

et le code utilisant cette fonction:

```
(1..3).each { |v|
  begin
    exceptionFunc(v)
  rescue => exc
    puts "Exception: #{exc.message}"
    exc.backtrace.each { |e| puts "\#{e}" }
  ensure
    puts "exécuté, quoi qu'il arrive."
  end
}
```

Analyse de logs de serveur HTTP

Pour terminer cette première partie sur Ruby, nous allons rapidement commenter un exemple pratique: un script qui lit un fichier de *logs* d'accès à un serveur HTTP et qui les écrit sous forme de fichier HTML.

Le format du fichier de *logs* est le suivant: nom ou adresse IP du client, utilisateur, mot de passe, date et heure de l'accès, requête, code réponse du serveur et taille des données transmises. C'est le format dit «*common*» d'Apache, utilisé par la plupart des serveurs HTTP. Le but du programme est d'afficher deux tableaux, un pour les

requêtes qui «se sont bien passées» (code retour inférieur à 400) et un autre pour les requêtes erronées (code retour supérieur ou égal à 400). Le format d'affichage est le suivant:

- Nom du client 1 (ou no IP)
- Fichier accédé n° 1 (nombre d'accès)
- Fichier accédé n° 2 (nombre d'accès)
- Nom du client 2 (ou n° IP)

Etc.

Le code source du programme est fourni sur le site web de *Programmez!* Son fonctionnement est le suivant:

1. Pour chaque ligne du fichier de *logs*, on extrait et on renvoie: le nom du client, la méthode d'accès, le fichier demandé, le code retour et la taille des données sous forme de tableau (fonction `splitLigne`).
2. Selon le code de retour, on range ce tableau dans `acces_ok` ou `acces_ko`.
3. Chaque tableau est trié (fonction `sort!`) d'abord par nom de client, puis par nom de fichier accédé (fonction de comparaison `cmp`).
4. Chaque tableau est alors affiché en HTML (fonction `affiche`).

N.B.: ce programme sert d'exemple aux fonctionnalités évoquées au long de cet article: il n'est ni représentatif de la «bonne» manière de coder en Ruby (la définition de quelques classes aurait été la bienvenue) ni optimisé.

Pour conclure

Ruby est un langage très riche de par ses différentes facettes: langage de script, langage fonctionnel, langage impératif, langage orienté objet et le plus souvent tout à la fois.

Ce mois-ci, nous avons donné suffisamment d'informations pour que tout lecteur intéressé puisse démarrer en Ruby sans trop de problèmes.

Nous sommes cependant passés à côté de nombreuses fonctionnalités dont les principales sont la programmation objet et l'utilisation des bibliothèques standard. Ce sera l'objet des prochains articles. ■

Quand il faut **montrer** patte blanche : **RSA/MD5**

Diverses méthodes sont utilisées pour sécuriser vos échanges sur Internet. Ce mois-ci, nous vous proposons de découvrir deux algorithmes : RSA, reposant sur la factorisation des grands nombres et MD5 permettant de calculer l'empreinte d'un document.

Par Laurent FOYNARD et Christophe BABAYOU, Ingénieurs chez UPLOG.

contact@uplog.com

Le RSA

RSA est un cryptosystème à clé publique basé sur la factorisation des grands nombres.

Cet algorithme a été nommé après que Ronald Rivest, Adi Shamir et Leonard Adelman ont publié le procédé en avril 1977. À ce jour, il a été très largement employé dans les applications de commerce électronique sur Internet. Il est aussi utilisé par les navigateurs Netscape Navigator et Internet Explorer pour l'implémentation de la couche SSL (*Secure Sockets Layer*).

Cette couche logicielle est employée pour les transactions de données confidentielles sur la toile. Elle met en œuvre toute une procédure d'authentification faisant intervenir le présent algorithme. Mastercard et VISA utilisent le RSA dans les protocoles de transactions électroniques sécurisées (SET) pour les diverses opérations liées à l'usage de la carte bancaire.

Clé publique et signature électronique

Le RSA est une implémentation du concept générique d'algorithme à clé publique. Ce dernier permet à deux parties qui ne se sont jamais rencontrées de communiquer dans un environnement fiable et sécurisé. Il faut savoir qu'aujourd'hui

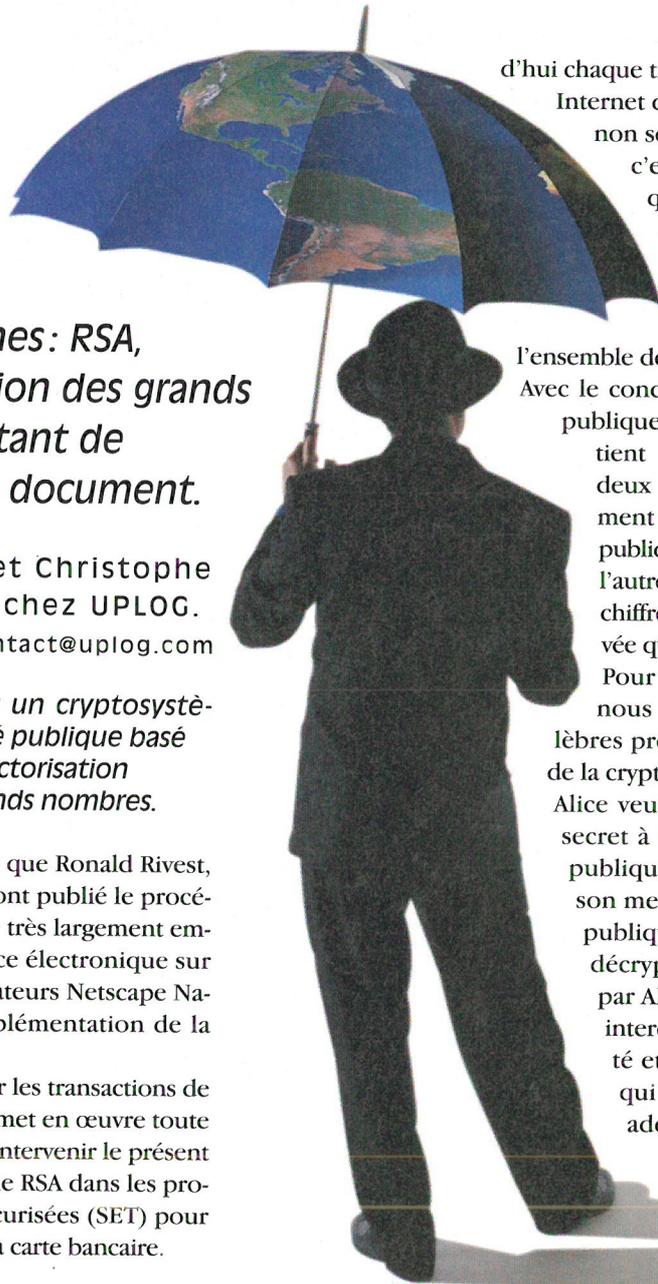
chaque transaction effectuée sur Internet dans un environnement non sécurisé transite en clair, c'est-à-dire que n'importe qui équipé d'un bon sniffeur (logiciel qui capture les trames circulant sur la toile) est capable de connaître

l'ensemble de vos transactions.

Avec le concept d'algorithme à clé publique, chaque utilisateur détient deux clés. L'une des deux est destinée au chiffrement de l'information (la clé publique qui est distribuée) et l'autre est employée pour déchiffrer le message (la clé privée qui doit rester secrète).

Pour illustrer ce principe, nous allons utiliser les célèbres protagonistes du monde de la cryptographie : Alice et Bob. Alice veut envoyer un message secret à Bob, elle utilise la clé publique de Bob pour crypter son message. Parce que la clé publique de Bob ne peut pas décrypter le message chiffré par Alice, personne ne peut intercepter le message crypté et le decoder. Seul Bob qui possède la clé privée adéquate est capable de déchiffrer le message

d'Alice. Finalement Alice n'a pas eu besoin de rencontrer Bob pour lui faire parvenir un message crypté de la plus haute confidentialité qui contient souvent des *logins* ou des mots de passe. Ceci représente une grande avancée face aux méthodes anciennes qui nécessitaient un échange préalable de clés privées entre les deux parties.



Ce système peut aussi être utilisé pour que Bob soit sûr que l'émetteur du message est bien Alice. Si Alice veut signer son message, elle peut le crypter avec sa clé privée. Lorsque Bob reçoit le message crypté, lequel est supposé venir d'Alice, il utilise la clé publique d'Alice pour le décrypter.

Si un document en clair apparaît, alors Bob peut avoir l'assurance que l'émetteur correspond bien au tiers attendu, car la clé publique d'Alice déverrouille son message. Ce dernier ayant été chiffré par sa clé privée qu'elle seule connaît.

Bien évidemment, signé électroniquement, le contenu du message ne devient pas privé, car chaque personne détenant la clé publique d'Alice peut déchiffrer et lire son document qu'elle a crypté à l'aide de sa clé privée.

Alice peut envoyer un message privé signé à Bob en le cryptant dans un premier temps avec la clé publique de Bob (à ce moment précis, seul Bob peut déchiffrer le message à l'aide de sa clé privée), puis dans un second temps en cryptant le résultat précédent avec sa clé privée, ce qui constitue la signature.

Toute personne recevant le message d'Alice peut déchiffrer la seconde étape du cryptage, mais seul Bob pourra décrypter la première étape du chiffrement et lire le message en clair.

Méthodes mathématiques

Les techniques mathématiques utilisées pour le cryptage transforment un message (qui est représenté comme une suite de nombres) en un texte crypté.

Certaines opérations mathématiques ont une dénomination de fonction à sens unique, ce qui convient parfaitement aux tâches requises par le RSA. Une fonction à sens unique est facile à calculer dans un sens, mais l'opération en sens inverse se révèle être une opération d'une très grande difficulté à réaliser.

Comme exemple simple, prenons le cas d'une assemblée de personnes, et demandons-leur de calculer de tête le carré de 23. Beaucoup vous répondront sans difficulté 529. Imaginez maintenant que vous demandiez à l'assemblée de calculer de tête la racine carrée de 529 (sans avoir eu connaissance de l'opération précédente). Une très grande majorité, pour ne pas dire la totalité n'y parviendra pas, ou du moins pas sans l'aide d'outils tels qu'une feuille de papier et un crayon. Le système RSA utilise des fonctions à sens unique de nature plus complexe telles que les opérations modulaires. Cette opération transforme le texte ou une partie du message en un bloc de texte chiffré illisible.

L'arithmétique modulaire est souvent appelée l'arithmétique de l'horloge, car les additions, soustractions, et autres opérations fonctionnent comme une horloge de type 12 heures. Par exemple, dans le mode 12 heures, 10 heures ajouté à 9 font 7 heures ($10h + 9h$ n'est pas égal à 19h mais à 7h). En fait, nous avons seulement retiré 12h (ou un multiple si nécessaire) pour obtenir le résultat.

$$7 = (10 + 9) \text{ mod } 12$$

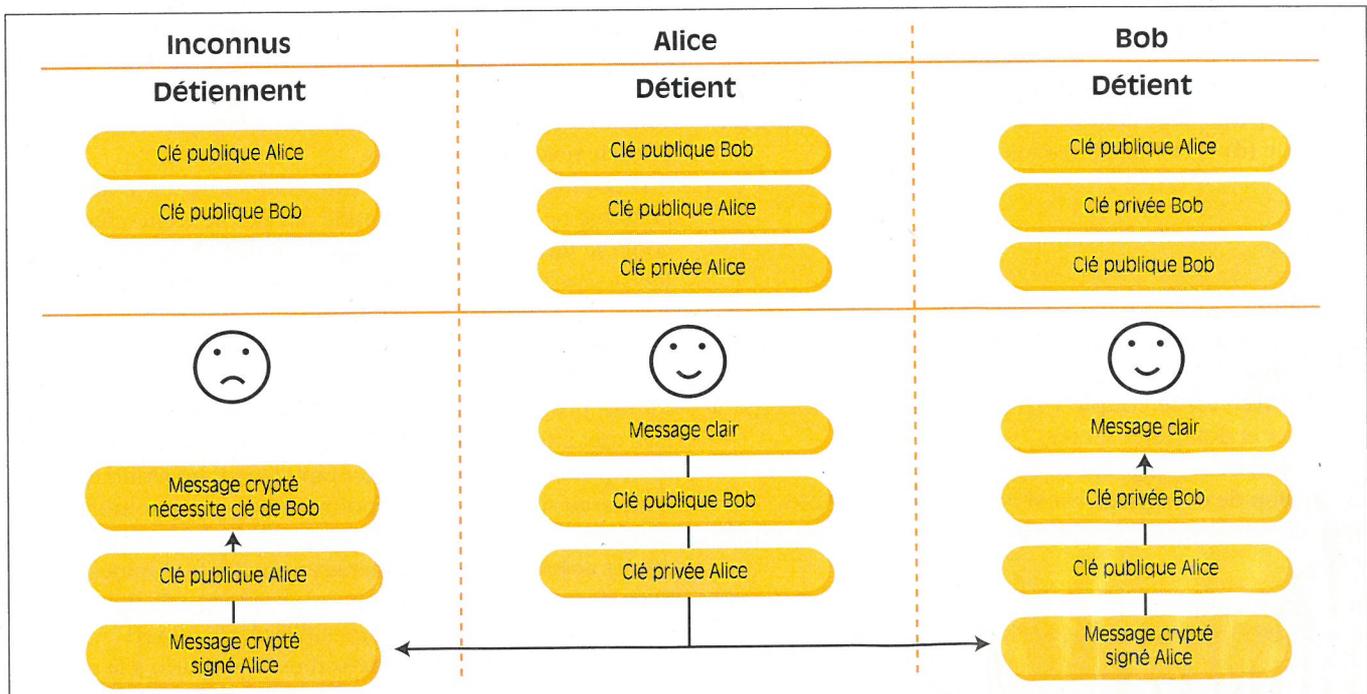
$$7 = 19 \text{ mod } 12$$

Ce processus est souvent appelé réduction modulaire. En soustrayant le modulo (ou un multiple du modulo) on obtient le nombre réduit.

Le RSA utilise des multiplications dans l'arithmétique modulaire. Plutôt que de multiplier un nombre par un autre nombre, le RSA multiplie un nombre (appelé la base) par lui-même un certain nombre de fois. Le nombre de fois que la base est multipliée par elle-même correspond à l'exposant :

$$16 = 2 * 2 * 2 * 2$$

$$16 = 2^4$$



> Les possibilités de lecture des messages en fonction des clés (publiques et privées) que détiennent les interlocuteurs.

Dans cet exemple, le nombre 2 correspond à la base, et le nombre 4 est l'exposant.

Dans la formule de cryptage du RSA, le message (qui est représenté par M) est multiplié par lui-même (e) fois (élévation de «M» à la puissance «e»), et le produit est divisé par le modulo «n». Le résultat de l'opération (souvent nommée puissance modulaire ou exponentiation modulaire) correspondant au texte chiffré.

$$C = M^e \text{ mod } n$$

Cette opération est extrêmement difficile à inverser lorsque le nombre est très grand.

Le décryptage se fait par l'opération suivante :

$$M = C^d \text{ mod } n$$

Le modulo (n) est un nombre construit à partir d'une multiplication de deux grands nombres premiers (p) et (q). Ils doivent avoir une longueur d'au moins 1024 bits chacun.

$$n = p * q$$

Pour calculer les exposants (d) et (e), il faut utiliser la formule suivante :

$$d = e^{-1} \text{ mod } ((p-1) (q-1))$$

Pour calculer la clé de cryptage, il est nécessaire de connaître les nombres (p) et (q) (appelés facteurs) utilisés pour calculer le modulo (n). Lorsque (n) est un nombre suffisamment grand, il devient impossible de retrouver ses deux facteurs premiers.

On peut diviser le RSA en trois étapes :

- la génération des clés, dans laquelle les deux facteurs premiers sont choisis et multipliés entre eux pour former le modulo (n). L'exposant de cryptage (e) est choisi de telle sorte qu'il soit inférieur à (n), et l'exposant de décryptage (d) est calculé en utilisant (e), (p) et (q).
- le cryptage, dans lequel le message (M) est élevé à la puissance (e) et réduit au modulo (n).
- le décryptage, dans lequel le texte crypté (C) est élevé à la puissance (d) et réduit au modulo (n). Exemple :

```
p = 61
q = 53
pq = 3233
e = 17 (est choisi de telle sorte qu'il soit inférieur au produit p*q)
d = 2753
```

La clé publique est le couple (3233, 17). C'est ce couple qui est distribué.

La clé privée est le couple (3233, 2753). Ce dernier est à conserver et doit rester secret.

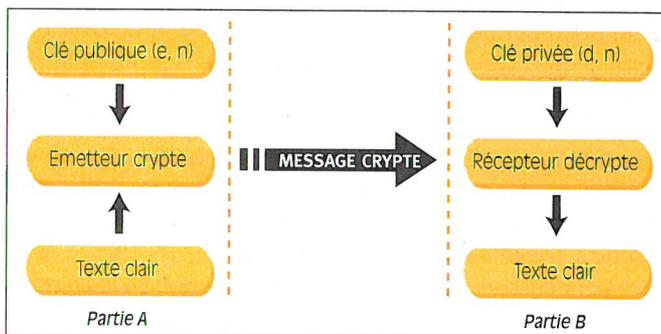
La fonction de cryptage est $C = (M^{17}) \text{ mod } 3233$.

La fonction de décryptage est $M = (C^{2753}) \text{ mod } 3233$.

Considérons la valeur 123 pour le message à crypter, on obtient :

```
C = (12317) mod 3233
C = 337587917446653715596592958817679803 mod 3233
C = 855
décryptons le texte chiffré 855 :
M = (8552753) mod 3233
```

```
M = {5043288895841606873442289912739446663145387836003550
9315554967564501 ... nous vous épargnons 7956 décimales ;)...
305946326827861270203356980346143245697021484375}
mod 3233
M = 123
```



> L'échange de message chiffré.

Le padding :

Le RSA est un algorithme de chiffrement travaillant sur des blocs de données. La taille de ces blocs est normalisée et dépend directement de la taille du modulo (n). Étant donné que la longueur du dernier bloc correspond rarement à la taille requise, il est nécessaire d'ajouter des bits de bourrage que l'on nomme couramment bits de *padding*.

Bien entendu, le *padding* se fait avant toute action de cryptage, et sera retiré après le décryptage.

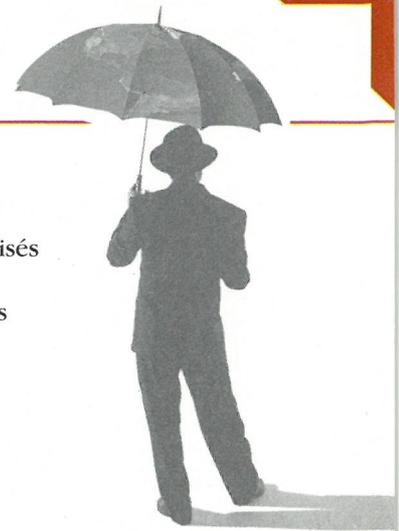
Le modèle appliqué au RSA est le suivant :

```
00 || BT || PS || 00 || D
```

- BT (*Blok Type*) est codé sur un octet. «00» ou «01» indiquerait un cryptage à clé privée. Comme il s'agit d'un cryptage à clé publique, sa valeur sera «02».
- PS (*Padding String*) a une taille variable et correspond au *padding* à proprement parler. Pour des algorithmes à clé privée, il peut prendre la forme d'une suite «00» ou de «FF». En ce qui concerne les algorithmes à clé publique, il doit prendre la forme d'une suite de chiffres aléatoires tout en excluant la valeur zéro (cette valeur permettant d'indiquer la séparation entre le texte et le *padding*).
- D (*Datas*) est le bloc de texte clair à crypter. Pour que le *padding* ne puisse être identifié par des tiers cherchant à décoder le message crypté, sa taille minimale doit être de 11 octets, le *Padding String* le plus petit sera de 8 octets.

Pour conclure

L'atout principal du RSA est qu'il est très sûr et très résistant face aux attaques actuelles. À ce jour, aucune machine n'est assez puissante pour factoriser rapidement un grand nombre. Il faudrait un an de calcul pour factoriser un nombre à 150 chiffres à l'aide de machines dont le coût exorbitant freinerait les ardeurs des plus téméraires (quelques dizaines de millions de francs), et 1000 ans pour un nombre de 200 chiffres. Le budget s'élevant à plusieurs milliards de francs ! Son inconvénient est sa lenteur d'exécution liée à la manipulation de très grands entiers. Par conséquent, on restreindra son utilisation à la transaction de données de petites tailles.



Le MD5

L'empreinte digitale humaine est unique.

Le MD5 applique ce principe à tout document à sécuriser.

Une fonction de hachage opère sur un message d'une longueur quelconque. L'objectif est de générer une empreinte unique pour un document donné. En d'autres termes, deux messages différents ne doivent pas avoir la même empreinte. Par ailleurs, l'empreinte doit être facile à retrouver à partir d'un message. En revanche, elle ne doit pas permettre la reconstitution du message originel. Les algorithmes de hachage à sens unique sont employés dans les procédés d'authentification.

La longueur des empreintes est passée à 160 bits, cette taille permet un minimum de résistance puisqu'il faudrait un calcul de 280 empreintes aléatoires avant de trouver la bonne. Du point de vue mathématique, les fonctions de hachage à sens unique sont inspirées des fonctions de compression. Elles travaillent avec les empreintes calculées à partir des blocs précédents mixés avec un bloc provenant du message.

Présentation de l'algorithme :

Le MD5 (*Message Digest* version 5) est une fonction permettant de calculer une « empreinte digitale » d'un document. Si un seul caractère du message change, la fonction de hachage retourne une valeur complètement différente. Cet algorithme est utilisé dans les applications nécessitant une signature électronique. Ce dernier est mixé avec un algorithme de cryptage à clé privée.

La procédure de hachage se décompose selon les cinq étapes suivantes :

Ajout des bits de padding :

Le message est étendu de sorte que sa taille en bits soit congrue à 448 modulo 512 (c'est-à-dire que la taille du message en bits divisée par 512 donne le même reste que 448 divisé par 512, soit un reste de 448).

Pour un message ayant une taille congrue à 448 modulo 512, on effectue un *padding* qui est systématique. Ce dernier est effectué comme suit : un bit à « 1 » est ajouté au message, puis une série de bits à « 0 » sont concaténés jusqu'à ce que la taille du message devienne congrue à 448 modulo 512.

Ajout de la longueur :

Une représentation de la longueur sur 64 bits est ajoutée à la fin du message étendu. Cette longueur correspond au nombre de bits du message sans le *padding*. Si la taille du message dépasse la capacité de 64 bits, on la tronque en ne conservant que les 64 bits de poids faible. L'ajout de la taille s'effectue selon la norme Little Endian (cf. *Programmez!* n°27). Nous obtenons alors un message ayant une taille multiple de 512 bits, et par conséquent, un multiple de 16 mots de 32 bits.

Initialisation des buffers MD :

Quatre *buffers* de 32 bits sont utilisés pour calculer le *Message Digest*.

Ces quatre registres sont initialisés de la manière suivante (toujours en Little Endian) :

- registre A : 67452301
- registre B : efc dab89
- registre C : 98badcfe
- registre D : 10325476

Calcul de l'empreinte par blocs de 16 mots :

Nous définissons 4 fonctions auxiliaires

prenant en entrée 3 mots et générant un mot en sortie :

```
F(X,Y,Z) = X and Y or not(X) and Z
G(X,Y,Z) = X and Z or Y not(Z)
H(X,Y,Z) = X xor Y xor Z
I(X,Y,Z) = Y xor (X or not(Z))
```

Il faut suivre les procédures suivantes :

```
/* On travaille sur chaque blocs de 16 mots */
For i = 0 to N/16-1 do
/* Copie le bloc i dans X. */
For j = 0 to 15 do
Set X[j] to M[j*16+i].
end /* fin de la boucle pour j */
/* Sauvegarder A en tant que AA, B en tant que BB, C en tant que CC,
et D en tant que DD. */
AA = A
BB = B
CC = C
DD = D

/* Ronde 1. */
/* Afin de simplifier la lecture de ce pseudo code, nous proposons que
[abcd k s i] désigne l'opération : a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s).
le symbole "<<< s" est une rotation circulaire à gauche de "s" bits*/

/* Effectue les 16 opérations. */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

/* Ronde 2. */
/* [abcd k s i] désigne l'opération :
a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
/* Effectue les 16 opérations. */
[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

/* Ronde 3. */
/* [abcd k s i] désigne l'opération :
a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
/* Effectue les 16 opérations. */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

/* Ronde 4. */
```

```

/* [abcd k s t] désigne l'opération :
a = b + ((a + l(b,c,d) + X[k] + T[i]) <<< s). */
/* Effectue les 16 opérations. */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]
[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

/* Ensuite on effectue les additions suivantes. (Ce qui revient à incrémenter
chacun des 4 registres par la valeur qu'ils avaient avant les calculs
effectués par ce bloc.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD
end /* fin de la boucle de i */
    
```

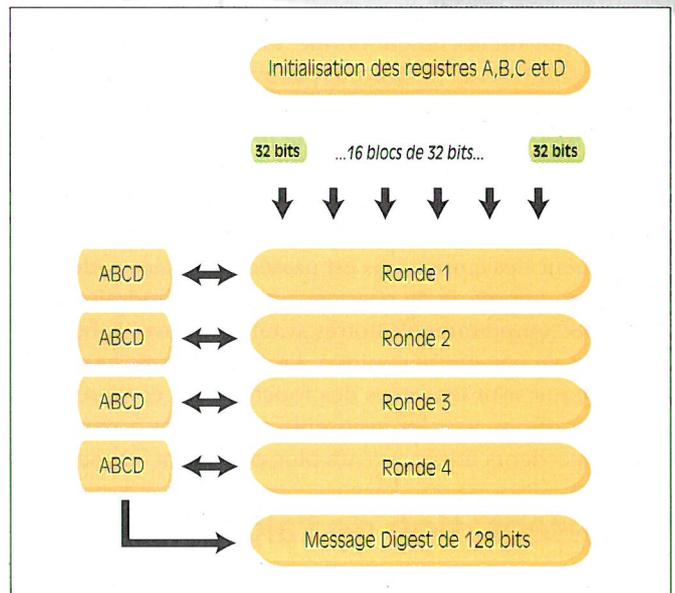
Ce que l'on obtient en sortie :

L'empreinte produite en sortie provient des 4 registres A, B, C et D. On concatène ces 4 registres en commençant par l'octet de poids faible de A, et en terminant par l'octet de poids fort de D.

Conclusion :

Le MD5 est un algorithme de hachage à sens unique simple à implémenter, rapide et fiable. Il est utilisé dans le monde Linux pour vérifier le bon déroulement des transferts de fichiers. En comparant la signature

de référence à la signature générée à partir du fichier transféré, il est possible de vérifier l'intégrité des données. MD5 est aussi utilisé dans le célèbre logiciel de chiffrement PGP (*Pretty Good Privacy*) pour vérifier l'intégrité d'une clé publique. ■



> Calcul de l'empreinte d'un fichier.

Choisissez la formule gagnante

Compatibles PC Magazine évolue pour offrir aux utilisateurs exigeants que vous êtes l'information que vous attendez.

- ✓ PLUS D'INFORMATIONS
- ✓ PLUS DE TESTS
- ✓ PLUS D'INTERNET
- ✓ PLUS D'UPGRADE
- ✓ PLUS DE PRATIQUE

Avec chaque numéro un CD-ROM exclusif avec des programmes complets, des utilitaires, des sharewares, des démos jouables... pour aller plus loin avec votre PC.



Compatibles **PCmagazine** : le magazine des utilisateurs avancés

ACTUELLEMENT EN VENTE CHEZ VOTRE MARCHAND DE JOURNAUX

■ Siticom Group

propose des stages de fin d'études (pour les élèves ingénieurs en 3^e année) d'une durée de 5 à 6 mois et des stages d'un an (pour des étudiants de 2^e année). Lors de ces stages, généralement préalables à une embauche, les stagiaires sont intégrés en tant que consultants au sein d'une équipe projets (en accord avec les clients) tout en réservant environ 30% de leur temps pour travailler sur un état de l'art qui leur servira de support pour leur mémoire de stage. Quelques exemples de missions : modélisation du trafic Internet d'un ISP pour le dimensionnement de son infrastructure ; schéma directeur d'un opérateur entrant ; étude d'intégration du trafic IP sur une infrastructure d'opérateurs de transit ; appel d'offres de service WAN IP pour un réseau international.

www.siticom.com

Agenda

■ La 43^e édition de Prosearch se déroulera les 21 et 22 mars prochain au CNIT. L'édition de janvier rassemblait 120 entreprises qui proposaient plus de 15 000 postes
Renseignements : www.prosearch.com

■ La prochaine édition parisienne des « jeudis de l'informatique » se déroulera le 8 mars, de 16h à 23 heures, au Palais des Congrès - Porte Maillot et réunira environ 70 exposants. Des manifestations similaires sont désormais organisées en région : le 15 mars à Grenoble, le 22 mars à Nantes et le 5 avril à Nice. Renseignements et offres d'emplois : www.lesjeudis.com

Recrutement

Nouvelles du front

• **Aurora**, éditeur et intégrateur de solutions e-business en environnement *Open Source*, a lancé un plan de recrutement de 60 personnes, notamment pour son département R&D. Le site www.aurora-linux.com permet de consulter toutes les offres d'emploi, d'envoyer un CV, et de découvrir la société et son activité à travers une présentation multimedia synchronisée constituée d'interviews de ses collaborateurs et réalisée avec son produit.

• *Open Source* toujours : **Asyres** recherche 14 administrateurs systèmes et réseaux et 11 développeurs. Ces postes sont ouverts aux Bac +2. www.asyres.fr

• **Coframi**, spécialisée dans l'informatique industrielle et embarquée, innove en organisant des journées de recrutement dans les stades. Après le Parc des Princes et le stadium de Toulouse, la société songe au Stade de France. Ces journées,

qui souhaitent mêler les côtés sportifs et festifs interviennent peu après des événements d'envergure. Les échanges entre recruteurs et candidats se font dans les gradins et sont suivis d'un cocktail. **Coframi** prévoit d'embaucher 400 personnes en 2001. www.coframi.fr

• **Qualiope**, agence de mesure de la qualité d'Internet et des télécoms, souhaite doubler ses effectifs d'ici à la fin 2001 en recrutant une quarantaine de personnes, dont 30% à l'international (Allemagne, Royaume-Uni, Hollande). En France, 40% des postes à pourvoir sont sur Aix et 60% sur Nanterre. Profils recherchés : ingénieurs développement bases de données, ingénieurs développement Internet, ingénieurs développement téléphonie, direction des opérations techniques, chef de projets e-Qmetrix, techniciens support filiale, techniciens d'exploitation,

webmasters.

www.qualiope.com

• La SSII **Valoris** prévoit d'embaucher 500 personnes cette année. Elle met en avant sa politique des ressources humaines basée sur « la fidélisation de ses cadres qui repose sur une culture d'entreprise originale, savante combinaison de l'ordre et du désordre, et qui laisse une large part à l'initiative de chaque collaborateur du groupe », ainsi qu'une politique de stock-options. www.valoris.com

• La SSII **Antall** recherche 260 nouveaux collaborateurs, dont un quart de jeunes diplômés. www.antall.com

• La filiale française de l'éditeur d'ERP **PeopleSoft** prévoit d'embaucher une centaine de personnes en 2001. Des commerciaux, mais aussi des consultants (3 à 5 années d'expérience, Bac +4 et Bac +5). www.peopleSoft.fr

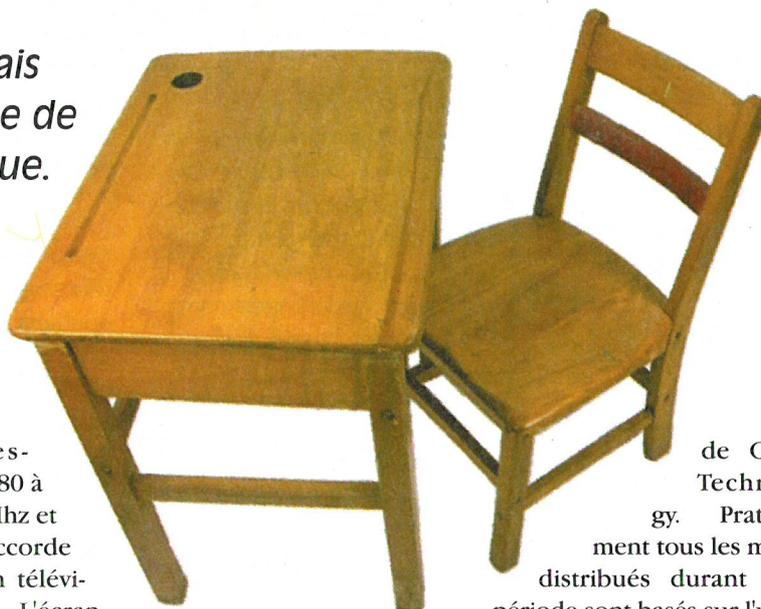
Reconversion

Transiciel propose des CDI avec une formation de sept semaines au sein de ses « pépinières formation » aux Bac+2 et Bac+4/5 de filières scientifiques voulant faire carrière dans l'informatique, et pour lesquelles la SSII organise des sessions sur mesure, plusieurs fois par an. Deux sessions ont débuté en janvier sur l'environnement grands systèmes MVS, l'une (Bac +2) pour les métiers de techniciens d'exploitation au sein du département « Infogérance et administration de systèmes et de réseaux », l'autre (bac+4/5) en vue de devenir développeur d'applications pour le département « Intégration de systèmes ». Les offres de candidature sont à adresser par mail à drh@transiciel.com. ●

Basic... Élémentaire, mon cher Watson!

Basic ne signifie pas simple, mais l'est quand même. Il est la base de l'apprentissage de l'informatique. Le B-A-BA (sic) des langages.

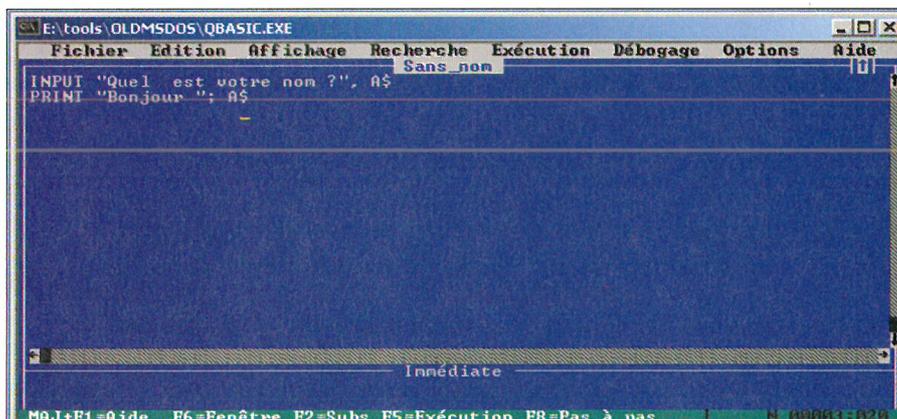
Jean-Marc Quéré
apiend@club-internet.fr



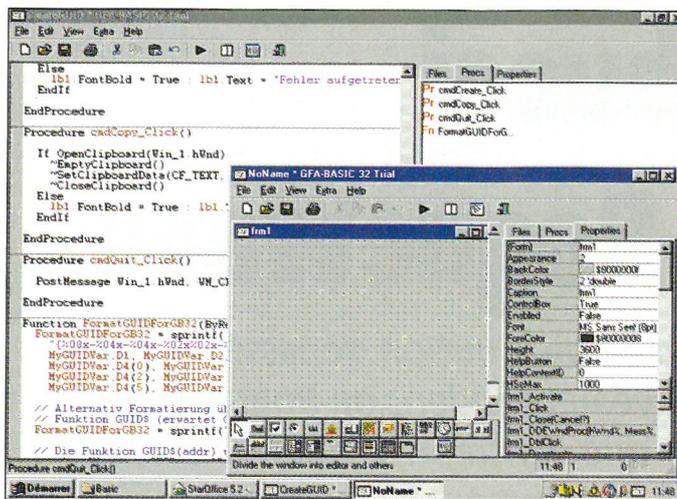
La brève histoire de l'informatique retient deux événements majeurs pour l'année 1964 : le lancement de la famille d'ordinateurs IBM System/360 et l'apparition d'un nouveau langage très simple destiné à l'apprentissage de l'informatique. Réalisé par Thomas Kurtz et John Kemeny au célèbre et réputé *Dartmouth College* (États-Unis), le *Beginners All-purpose Symbolic Instruction Code* était avant tout destiné aux étudiants. Pour situer les faits, IBM fabriqua la première disquette en 1967. La principale caractéristique du langage est l'usage de numéros de lignes pour gérer la séquence des traitements avec de nombreux sauts, des renvois, etc. Il s'est largement répandu depuis et a beaucoup évolué. Les étapes les plus marquantes sont principalement liées à l'essor des équipements informatiques. Les premiers ordinateurs destinés au « grand public » (en fait, des amateurs avertis) contenaient le Basic en Rom et lors de leur mise sous tension proposaient pour seule invite celle de l'interprète de commandes. Les modèles les plus marquants auxquels nous devons la démocratisation de la micro-informatique sont les ZX80 et 81 (respectivement avril 80 et mars 81). L'ordinateur se compose d'un clavier de 40 touches, d'un port cassette de 300 bauds, d'un

processeur Z80 à 3,25 Mhz et se raccorde sur un téléviseur. L'écran noir et blanc de 24 lignes de 32 caractères offrent également un mode graphique de 48 par 64 pixels. La mémoire disponible pour les applications est de 1 Ko extensible à 32 Ko. Ces microordinateurs se sont vendus à des millions d'exemplaires (500 000 unités la première année pour le ZX81) et étaient livrés avec un manuel, « *ZX81 Basic Programming* » par Steven Vickers. En parallèle de 1971 à 1980 se développent les microprocesseurs 8080 d'Intel, le 6800 de Motorola et le 6502

de C-Mos Technology. Pratiquement tous les micros distribués durant cette période sont basés sur l'un ou l'autre des processeurs évoqués et intègrent le Basic dans leur Rom (version spécifique à chaque équipement). Paul Allen et Bill Gates, futurs fondateurs de Microsoft, sont déjà sur les rangs, dès 1975 avec l'Altair. La presse informatique fait son apparition. Elle commente l'actualité des Atari XL, Oric, Spectrum TO7, Vic20, ZX, etc., et offre en ses colonnes des codes sources de programmes réalisés en Basic (en général par les lecteurs). Courant 1985, l'arrivée d'une nouvelle génération d'ordi-



QuickBasic.



GFABasic, le retour ?

nateurs (AtariST, Amiga, Amstrad, ...) apporte des évolutions significatives : le Basic devient procédural et structuré. Les capacités sonores, graphiques et de stockage introduisent des instructions utiles à leur gestion. Chaque fabricant, propose ses propres extensions incompatibles avec celles des autres éditeurs. Proposé initialement sur Atari XL, le TurboBasic de Franck Ostrowski devient le GFABasic et est complété d'un compilateur. Il s'impose sur Atari et Amiga comme le langage de développement par excellence. Le monde PC poursuit son chemin avec l'arrivée de QuickBasic (également doté d'un compilateur), successeur de Basic. L'apparition de Windows et l'intégration de nouvelles méthodes de développement des interfaces utilisateurs (par cliquer-déposer de composants) se sont concrétisées par l'apparition de Visual-Basic (Microsoft) et la programmation événementielle. L'environnement proposé était tellement novateur qu'il est devenu une norme (RAD) et est maintenant disponible pour l'ensemble des plates-formes de développement.

En pratique

Le Basic est le plus accessible de tous les langages de programmation. Même si vous n'avez jamais programmé auparavant, vous pouvez apprendre le Basic. Vous devez seulement disposer d'un interprète ou d'un compilateur. Avant de vous lancer dans un investissement significatif (VisualBasic?), vous

pouvez essayer d'utiliser une version largement distribuée : *Quickbasic*. Les distributions de Windows comportent cet exécutable (pour DOS) dans un répertoire sur le cédérom d'installation. Dans le cas de Windows 98, il se cache dans «tools\oldmsdos». Le programme se nomme «QBASIC» et se lance à partir de l'interprète de commande MS-DOS ou plus simplement en cliquant sur son icône. Vous pouvez alors saisir vos commandes à l'intérieur de la fenêtre et en provoquer l'exécution par le menu de même nom (ou shift-f5). À titre d'exemple, le programme ci-dessous vous demande votre nom (stocké dans A\$) et affiche «Bonjour» suivi de votre nom.

```
INPUT " Quel est votre nom ? ",A$
PRINT " Bonjour ";A$
```

Un futur ?

Le Basic a permis à une multitude d'informaticiens professionnels et amateurs de découvrir puis de s'investir dans la micro-informatique. Longtemps décrié pour une programmation de type «spaghetti», le langage a évolué en intégrant progressivement de nouvelles notions imposant un minimum de méthode : procédure, programmation événementielle, etc. Toutefois, aucune des ces technologies ne lui est propre. La simplicité de ce langage a poussé Microsoft à l'introduire dans ses applications bureautiques sous la forme de VBA (Visual Basic pour Appli-

```
10 PRINT "QUEL EST VOTRE NOM"
20 INPUT A$
30 PRINT "BONJOUR "; A$
```

À l'heure du ZX81

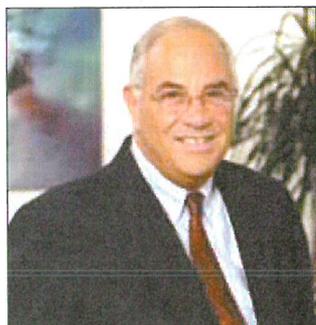


Le ZX81...

cation). Peu d'utilisateurs se sont intéressés à ces nouvelles possibilités. Les programmeurs professionnels n'ont pas trouvé d'intérêt à la solution proposée et préfèrent intégrer les applications bureautiques à leur développement *via* COM, OLE, etc., que l'inverse. Les usagers de traitements de texte et autres tableurs ne souhaitent pas s'engager dans ce type de démarche et préfèrent rester du côté «consommateur» de l'outil. Le déclin du Basic est lié au désengagement des possesseurs et des utilisateurs de micro-ordinateurs qui ne s'intéressent plus ou n'ont pas la capacité de s'intéresser à la programmation : l'ordinateur n'est pas destiné exclusivement aux informaticiens. Le langage s'éteint à petit feu, malgré les tentatives des éditeurs de créer un «nouveau basic». La société GFA Software Technologie (<http://www.gfasoft.gfa.net>) tente de recouvrer sa splendeur passée en proposant une nouvelle version de son célèbre GFABasic pour PC en 32 bits (la précédente version 16 bits pour Windows 3.x avait fait un flop).

Chronique d'une mort annoncée

Le Basic a connu ses heures de gloire et est assurément l'un des vecteurs de la démocratisation de la micro-informatique. Hélas, actuellement, le marché orienté vers les utilisateurs finaux réserve la programmation à une rare élite : quelques amateurs avertis et les professionnels. Ces derniers l'ont abandonné depuis longtemps pour adopter d'autres environnements (à l'exception de VisualBasic qui conserve ses adeptes). Côté formation, les nouvelles technologies ont introduit des concepts qui sont plus facilement manipulables avec d'autres langages mieux adaptés : Perl, PHP, Java, etc. En effet, la nouvelle population de «programmeurs» issue de l'univers web opte d'avantage pour ce type de solution. Le Basic se meurt, tout fout le camp! ■



Jack Iacobucci surfe sur RogueWave

Un marché déclinant, le C++, un modèle de distribution dépassé, le logiciel packagé et pourtant Jack Iacobucci, nouveau président de Rogue Wave Software court le monde en expliquant que l'entreprise va doubler son chiffre d'affaires dans les deux ans. Voyons comment.

Stéphane Larcher

Vous avez démarré cette aventure il y a 8 mois, dans quelles conditions ?

Quand on m'a contacté, j'ai rencontré des gens merveilleux et intelligents, une base de clients comme toute entreprise en rêverait. Et plus de 30 M\$ de *cash* à la banque. Cette société était un peu comme une belle endormie. Si vous ajoutez que j'aime les défis, vous avez la réponse.

Le marché du C++ ne décline-t-il pas ?

Si, vraisemblablement. Mais ce n'est pas le plus important. Tout d'abord, ce marché va encore vivre pendant au moins une dizaine d'années. Nous avons des opportunités de croissance internationale énormes. Enfin, le langage C++ et nos propres composants sont utilisés pour bâtir des applications stratégiques, et c'est de cette manière que nous pouvons envisager une croissance à deux chiffres pour les prochaines années.

Soyez plus explicite sur le développement international.

Nous venons d'ouvrir des représentations en Chine et au Japon. Nous approchons d'autres marchés, comme l'Inde. C'est l'un des premiers axes de croissance. Le modèle économique que nous utilisons depuis la création de l'entreprise (à savoir la distribution de logiciels packagés *via* la distribution) aura un effet mécanique.

Et ensuite ?

Le second axe de croissance consiste à devenir plus une société impliquée

dans la fourniture de solutions complètes orientées objets (*Large Scale objects solution*) plutôt qu'un simple distributeur de composants comme nous sommes perçus aujourd'hui. Le changement est le suivant : dans le passé, nous distribuions nos logiciels, nos composants, nos bibliothèques, à charge pour les clients de les implémenter. Aujourd'hui, nous faisons la même chose, mais nous assurons en plus l'implémentation ou le conseil. Ce marché va représenter plus de 50 millions de dollars de chiffre d'affaires dans les deux ans. Il ne s'agit plus de transactions à 1 000 ou 2 000 dollars, mais de contrats portant sur des millions.

C'est un modèle ASP ?

Dans certains cas, cela va le devenir. Par exemple, nous venons de signer un contrat très important avec l'une des plus grandes banques mondiales. Pour ce projet, nous mettons en œuvre un modèle ASP et nous avons la volonté de décliner ce modèle auprès d'autres clients. L'ASP n'est pas une stratégie globale pour tous les secteurs d'activité. Pour la banque c'est clairement une demande des clients, et c'est pour nous l'occasion de développer un modèle économique très solide. Je trouve formidable d'être payé pour fournir de la technologie et être payé pour l'intégrer et la déployer. Nous n'avons pas de difficultés à vendre nos solutions, mais nous en avons pour trouver des gens pour les implémenter. C'est la raison pour laquelle le modèle ASP est très adapté.

Pourquoi ne pas le faire partout ?

Nous ne sommes pas une société assez importante. Mon objectif n'est pas de créer la société la plus importante en termes de personnel, mais d'avoir la meilleure rentabilité sur le nombre d'employés. Il me paraît plus important de nouer des partenariats avec des intégrateurs de systèmes. À travers l'offre *Large Scale objects solution*, nous développons les ventes directes. Nous voulons intervenir auprès des directeurs techniques, des chefs de projets et pas seulement auprès des développeurs. Nous pensons être bien placés pour montrer l'intérêt et la pertinence de nos produits dans leurs systèmes d'information. Pour la banque dont je parlais précédemment, l'objectif est d'accélérer les flux d'informations pour les collecter et les compiler avec d'autres données. Précédemment, cela prenait cinq jours et un seul maintenant. C'est un contrat très stratégique qui a mobilisé une centaine de personnes dont 70 développeurs.

Quelles sont les évolutions technologiques ?

Nous sommes engagés autour d'XML. Dans quelques semaines, nous annoncerons un partenariat stratégique avec Microsoft. Il y a plusieurs pistes.

Et l'Europe ?

L'Europe a réalisé l'année dernière un chiffre d'affaires de 16 millions de dollars en progression de 65 %. Cela reste un marché primordial. ■

Intelligence Artificielle

Faites l'Othello

2^e partie

Un jour, l'homme, ému, a réalisé qu'il était intelligent. Il a voulu en faire profiter ses machines. Il en a joué, il perd parfois quelques parties.



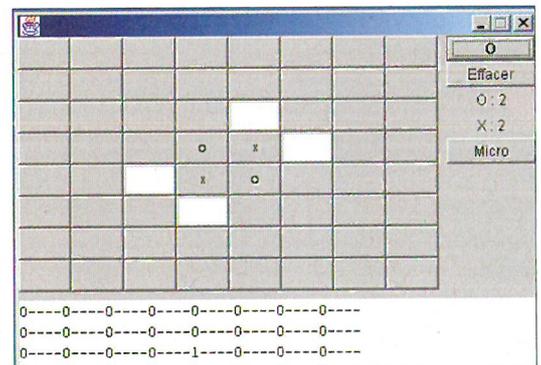
Sur votre CD-Rom

Jean-Marc Quéré
apiend@club-internet.fr

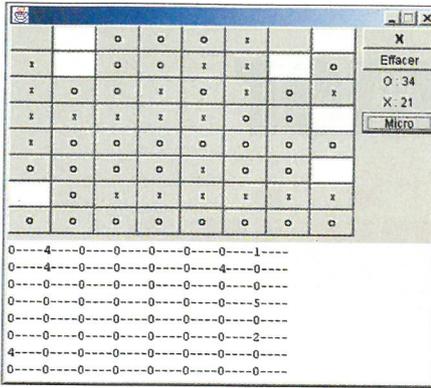
Toujours dans l'optique d'apporter un peu de cognitif à nos machines inanimées, l'approche proposée permet d'aborder les concepts évoqués lors de la première partie dans un nouveau cadre. Le défi consiste à permettre à l'ordinateur de jouer une partie d'Othello. La problématique reste la même : représenter, valoriser, estimer et jouer. Pour mémoire, le jeu d'Othello se compose d'un plateau de 64 cases sous la forme d'un carré de 8 par 8. Les pions comportent deux faces : une blanche et une noire. Dans le cadre de l'application, elles sont représentées par un « O » et un « X ». Au début de la partie, le centre du plateau est occupé par quatre pions formant un carré (de 2 par 2) dont la lecture de haut en bas et de gauche à droite donne la séquence « OXXO » (pions blancs et noirs disposés de façon alternée). L'objectif consiste à déposer un nouveau pion sur le plateau de sa couleur (blanche ou noire, donc deux joueurs). Le pion doit être placé à proximité immédiate d'un pion ou d'une ligne de pions adverses (de l'autre couleur) déjà limité(e) par un pion allié, (par exemple, la séquence suivante XOOO_). Tous les pions adverses situés entre le nouveau pion et un pion de même couleur constituant la ligne sont retournés, basculant dans la couleur du pion joué, (ce qui donne dans notre exemple : XOOO (+X) = XXXXX).

La démarche liée à un coup joué est double : constituer les plus grandes lignes possibles ou le plus grand nombre de lignes de sorte à obtenir un maximum de pions adverses. Ce dernier point définit l'objectif de la logique implémentée dans le cadre de cette nouvelle réalisation. À noter qu'en pratique, cette seule recherche de gain (nombre de pions pris à l'adversaire soustrait du nombre de pions qu'il est susceptible de capturer par la suite) n'est pas suffisante pour s'assurer la victoire. En effet, toutes les cases du plateau n'ont pas le même intérêt, la même valeur. Il est préférable à gain égal voire déficitaire de placer un pion sur une case « protégée ». Les cases protégées sont situées sur le bord du plateau et dans les quatre coins. En effet, il est plus difficile de retourner un pion situé sur le bord car il est moins exposé. Les quatre coins sont totalement imprenables. Non seulement, ces cases ont un intérêt défensif certain, mais en plus, elles sont stratégiques et permettent la capture de nombreux pions adverses. Le « bien jouer » consiste donc dans un premier temps à ne pas étaler ses pions, puis à adopter dès que possible, une stratégie offensive à partir d'une case « protégée ».

Laissons là – temporairement – ces notions de valeur de la case occupée pour nous intéresser dans un premier temps à l'aspect « gain » : capture du plus grand nombre de pions en s'assurant un minimum de perte. L'application « Othello » est composée de trois zones distinctes : en haut à gauche, le plateau de jeu (jPanelJeu) ; à droite, les boutons de commandes (Pion, Effacer,...) ; en bas, la partie message (jTextArea). Suite à l'activation de l'application les quatre pions de départ sont affichés. L'évaluation du plateau est affichée dans la zone message. À chaque case est associée une valeur indiquant le nombre de pions capturés si un pion y est déposé. L'estimation est calculée pour le type de pion sélectionné (O ou X). Vous pouvez utiliser le premier bouton en haut à droite pour chaque type de pion (le tour de jeu). Les données affichées sont



> Le plateau au départ.



> En cours de partie : micro contre micro.

actualisées en conséquence. Le plateau de jeu comporte des cases fond blanc. Seules celles-ci peuvent être jouées. Elles correspondent aux valeurs non nulles de l'évaluation. À ce stade, deux opportunités s'offrent à vous : vous jouez ou l'ordinateur joue. Si vous souhaitez jouer, cliquez directement sur le plateau la case choisie. Le pion s'affichera et les pions adverses seront «retournés». Dans le cas contraire, cliquez sur le bouton «micro». Le mécanisme d'évaluation des gains (et pertes) sera enclenché et la première solution de gain maximum sera jouée. Le processus peut être itéré au choix ce qui permet de faire jouer l'ordinateur contre lui-même ou d'intervenir à loisir pour modifier le cours de la partie.

Les méthodes

Côté programmation, deux classes sont définies : Jeu et Joueur. La classe Jeu comporte l'amorce de l'application (fonction **main**) et assure la gestion de l'interface utilisateur : affichage de la fenêtre et prise en charge des interactions avec l'opérateur. La classe Joueur comporte les méthodes relatives à la constitution d'une représentation du plateau, à son évaluation et à la «prise de décision». Son constructeur requiert l'indication du type de pion («O» ou «X») utile à la représentation (pions alliés / adverses). L'initialisation statique de la classe déclare et définit les valeurs associées à chaque direction : haut/gauche, haut, haut/droit, gauche, milieu, droit, bas/gauche, bas et bas/droite dans les deux axes (vrx, vry). La méthode **affiche** réactualise le plateau à partir d'une représentation. Elle est utilisée suite à un coup joué

pour indiquer la position du nouveau pion et l'état des pions retournés. La méthode **dump** affiche l'état évalué dans la zone message. Vous pouvez compléter cette dernière pour indiquer toute information que vous jugerez utile. **Evalue** retourne sous la forme d'un tableau d'entiers les valeurs associées à chaque case de la représentation communiquée. Les valeurs correspondent au nombre de pions pouvant être capturés si le coup est joué dans la case considérée (voir ci-dessus). L'ensemble des directions est parcouru à cet effet et le total cumulé. **Evalue** comporte deux implémentations : l'une effectue le traitement à partir d'une représentation, l'autre directement à partir du plateau (jPanelJeu) qu'elle représente préalablement à l'appel de sa première implémentation. **Represente** transpose les «X» et les «O» du plateau en 1 et -1 selon qu'ils s'agissent de pions alliés ou adverses. **Place** effectue l'ensemble des opérations relatives à la prise en charge d'un coup joué : ajout du pion et retournement des pions adverses.

Joue est la méthode-clé qu'il conviendra d'enrichir. Elle permet à l'ordinateur de simuler un joueur (médiocre en l'état). Elle se base sur une représentation du plateau pour évaluer le gain de chaque possibilité. Cette notion de gain correspond à la somme des pions capturés moins celle du meilleur coup adverse dans le cadre de ce coup joué. La méthode **somme** est une fonction récursive.

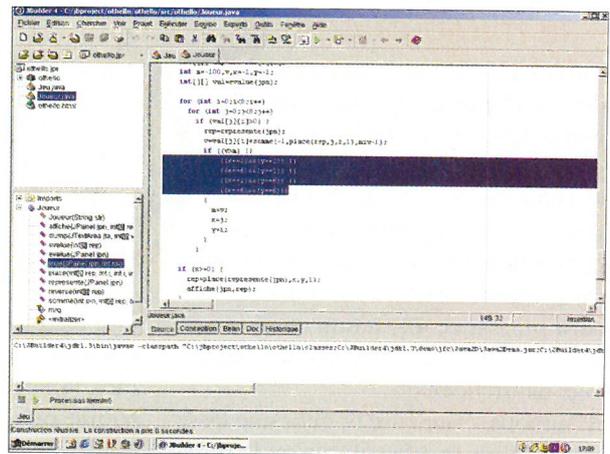
À chaque itération la représentation manipulée est «inversée» par la méthode **reverse** corrélée avec l'alternance des joueurs. Vous pouvez consulter les coordonnées et le gain associés à un coup joué par l'ordinateur en ajoutant la ligne :

```
« System.out.println(" -> "+x+ " , "+y+ " : "+i); »
```

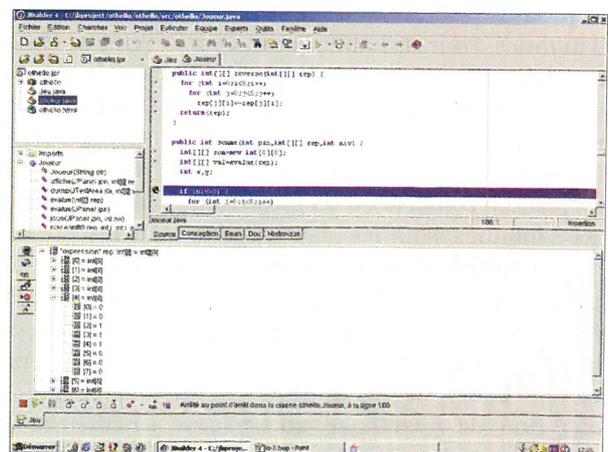
dans le bloc d'instruction associé au «**if(x>=0)**» de la méthode **joue**. Le niveau (1,2) d'évaluation peut être modifié dans la méthode **jButtonMicro_actionPerformed** lors de l'appel de

```
" new Joueur(...).joue(jPanelJeu, <1 ou 2>".
```

Parmi les améliorations possibles, le choix aléatoire d'une possibilité parmi plusieurs de même gain permet de diversifier le jeu (à bannir lors de l'étude du modèle : la simulation doit être reproductible). Notez que les cases en diagonale de chaque coin sont à éviter si possible, car elles offrent une position dominante à l'adversaire (coordonnées 1,1 / 6,1 / 1,6 et 6,6). L'ajout d'une valeur aux cases précédemment évoquées permet à l'ordinateur de ne pas céder trop facilement ces cases. En l'état, pour gagner, visez les cases au bord du plateau, pour perdre – et pourquoi pas ? – jouez au centre... ■



> Exclusion des cases «sensibles».



> La représentation en cours de traitement.

Tout programmer dans une cuisine

Aussi fort soit-il en VB, Java ou XML, le programmeur moyen fait moins le malin dans l'univers hostile à sa compréhension du monde (binaire) qu'est une cuisine normalement équipée. Heureusement, une fois de plus, Programmez ! est là pour vous aider à appréhender ces étranges objets.

Carole Pitras

Le basique : le micro-ondes

L'avantage du micro-ondes est qu'il sert à tout, son inconvénient étant que, comme tout généraliste, il est forcément limité pour ce qui est spécialisé. En fait, la plupart des micro-ondes comportent beaucoup trop d'options de programmation : les seules utiles sont puissance maximum et décongélation. Et le bouton temps de réchauffage naturellement.

Résumons ! Pour faire bouillir une tasse d'eau (pour le café soluble ou le thé) : 2 à 3 minutes suivant la puissance du four. Pour une cafetière entière : 6 à 8 minutes. Pour réchauffer une pizza : le micro-ondes n'est pas conseillé (ça ramollit la pizza), sauf si vous avez un mixte micro-ondes/grill auquel cas débrouillez vous, c'est une hérésie. Pour les plats en conserve ou du rayon frais : 2 à 3 minutes. Pour les plats surgelés : 4 à 5 minutes. Pour 400 grammes de purée du rayon frais : 3 à 4 minutes. Pour la même quantité de purée surgelée : 5 à 7 minutes.

Le truc bête à ne pas oublier : on ne met rien de métallique au « micro-ondes ». Les plats surgelés en barquette alu se mettent dans un plat spécial

micro-ondes. Et dans la mesure du possible on met un couvercle : sinon intérieur du four redécoré à nettoyer. Pas cool.

L'essentiel : la cafetière électrique

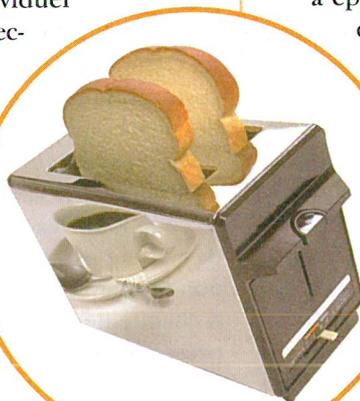
Si le distributeur de Coca est encore un équipement individuel rare, la cafetière électrique peut avantageusement pallier le manque d'excitant. Optez pour une cafetière programmable, bien sûr, qui fait réveil et vous permet d'ouvrir un œil avec l'odeur du café tout frais, ce qui est la façon la moins pire de commencer la journée. Elle sont programmables 24 heures à l'avance et s'arrêtent automatiquement après un laps de temps donné.

Il existe également des cafetières électriques type expresso que l'on nourrit avec des capsules pré-dosées. **Le plus :** un vrai expresso et pas de question à se poser sur le nombre de cuillères de café à mettre. **Le moins :** je n'en ai jamais vue de programmable.

Le truc bête à ne pas oublier : la veille au soir, mettre du café et de l'eau dans les réceptacles prévus à cet effet, et surtout, le récipient en dessous pour récupérer le café. Sinon se réveiller en ayant à éponger du café partout, brûlé qui plus est grâce à la plaque chauffante, et le tout sans avoir de café à boire, c'est une mauvaise façon de commencer la journée.

Le superflu : le four traditionnel

On m'a demandé de traiter le sujet, ce qui supposerait que les programmeurs font un minimum de cuisine évoluée. Je n'y crois pas trop, mais bon... Si vous n'avez pas un micro-ondes qui fait grill, le four traditionnel, ou à chaleur tournante vous servira essentiellement à faire réchauffer et griller une pizza ou un gratin



surgelés. Le seul truc à connaître est le bouton principal qui permet de passer du réchauffage au grill.

Pour une pizza surgelée, 12 à 15 minutes, pour une quiche ou assimilé entre 20 et 25 minutes, puis attendre 10/15 minutes avant de manger, «*achke chest a c h e m e n t chaud*».

Pour la température, je n'ai jamais rien compris au thermostat, la position 7 soit 200/220° a l'air d'être la plus utilisée,

mais demandez donc à votre mômman, avec toutes les précautions d'usage afin qu'elle ne fasse pas une attaque en pensant que vous vous lancez dans la cuisine.

Autre truc bête à ne pas oublier : la fonction grill est très utile, mais peut s'avérer dangereuse si on l'oublie. Le gratin de macaroni flambé sans alcool au grill du four, c'est pas mangeable, j'ai testé pour vous.

Utile : le lave-vaisselle

La aussi, il s'agit de science fiction : le programmeur lambda ne mangerait plus sa pizza à même le carton devant son écran mais dans une assiette, avec un couteau et une fourchette. Alors, traditionnellement, les assiettes et couverts se rangent en bas, les verres, bols et tasses à café en haut. On met le liquide adéquat ou, plus pratique, la pastille «3 en 1» lavage-rinçage-brillance et on tourne le bouton marche.

Le truc bête à ne pas oublier : n'y en a pas, c'est tout de même pas sorcier de se servir d'un lave-vaisselle ! Sinon faites la vaisselle à la main... Pour les cuisines minuscules, on trouve des lave-vaisselles intégrés à la cuisinière. (non, ce n'est pas du jpeg!).



> Le Screenfridge d'Electronux : seulement un prototype.

Le futile : tous les autres robots

Pour le seul petit déjeuner, il est possible d'utiliser au moins quatre robots : cafetière électrique, presse-fruit, bouilloire et grille pain. Pour le presse-fruit, inutile, il en existe des jus tout prêts en grande surface. La bouilloire : également inutile, l'eau chauffe très bien au micro-ondes.



> La cafetière programmable de Moulinex. Cindy Crawford n'est pas livrée avec.

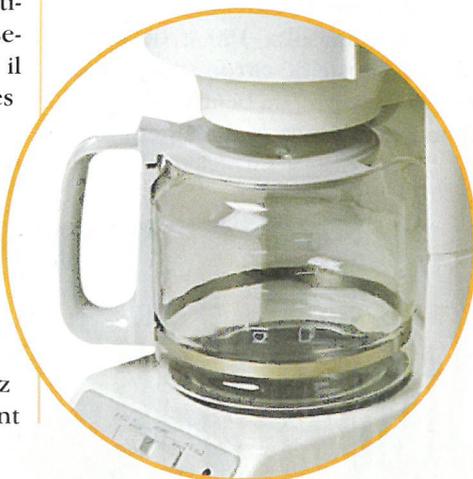
Le grille pain était fait pour les nuls en programmation : il fallait seulement décider si on aime les tartines très grillées (3 à 5 minutes suivant la puissance) ou peu grillées (2 à 3 minutes) et on le réglait une fois pour toute. Fi de cette belle époque ! Il existe maintenant des grille-pain très sophistiqués avec options viennoiseirie, baguette, toast, etc... Si il faut commencer à penser dès le réveil, c'est de la torture... Le mixer est fait pour les gens qui font un peu de cuisine, qui ont un bébé ou qui se sont récemment fait opérer des amygdales. Si vous êtes dans les deux premiers cas vous pouvez investir. Dans le troisième cas, optez pour les petits pots pendant

une semaine. Bon et puis, on ne va pas entrer dans le détail du reste. Si vous être un programmeur cuisinier vous connaissez déjà. Sinon, vous n'en avez rien à faire...

Dans un futur pas si lointain, la cuisine idéale sera peuplée d'électroménager intelligent : NCR avait innové en 1998 en présentant un prototype de micro-ondes qui permettait de surfer sur Internet, de lire les codes à barres afin de faire ses courses. Un prototype de réfrigérateur similaire, le «Screenfridge», a vu le jour en 1999 chez Electrolux. On nous promet en vrac des réfrigérateurs intelligents qui signalent les denrées périmées ou manquantes, qui feront les courses sur Internet à votre place ou encore, ceux qui, à partir des consommables stockés à l'intérieur, identifiés grâce à leurs codes à barres, proposeront des recettes. Le micro-ondes, à partir du code à barres, programmera seul le temps et le mode de cuisson. *Big Brother* n'est pas loin... Peut-être, ce jour là, les programmeurs feront-ils la cuisine, en Jini... Pour une vision de ce futur idyllique, voyez à :

http://www.webdo.ch/hebdo/hebdo_1999/hebdo_41/eco_conso_41.html.

L'article date un peu mais est très complet sur les cuisines du futur. Voir aussi nos actus de ce mois. ■



Scénario catastrophe

il y a un bug dans votre soirée, ne vous laissez pas envahir par le désespoir, comme souvent, Programmez! va vous sauver la mise. Merci qui?

François Denivet

C'est une catastrophe. Vous le saviez, cela devait arriver un jour ou l'autre, dans un moment d'égaré total, vous avez invité quelques amis à dîner. CHEZ VOUS! Vous, qui ne connaissez de la cuisine que le vague chemin qui mène au réfrigérateur. Vous, qui devez absolument terminer de tester cette superbe application dont vous êtes si fier et qui devrait vous faire connaître comme le Mozart du C++. Debout devant votre miroir, le regard plein de haine envers vous-même, vous vous traitez de bug. En moins de temps qu'il n'en faut à Bill Gates pour recevoir une tarte à la crème, vous avez composé votre menu de ce soir :

LA QUICHE AU SAUMON

Pêchez un saumon. Faites-le fumer. Débitez-le en fines tranches. Vous pouvez également vous procurer les 200 g de saumon fumé nécessaires chez cet aimable personnage appelé poissonnier, voire dans n'importe quelle épicerie fine. Pour mener à bien votre recette, vous aurez encore besoin de 3 œufs (inutile de chasser une poule, comme les champignons, les œufs se cueillent directement au sol); 150 g de crème fraîche, 1 zeste de citron (non traité), du sel, du poivre, du beurre, de la noix de muscade rapée et de l'aneth. Bien

qu'il soit possible de la confectionner vous-même, vous utiliserez une pâte brisée déjà prête, conditionnée en rouleau. Lisez le didacticiel de la pâte pour connaître le temps de cuisson. Beurrez et farinez un moule (24 cm de diamètre). Étendez-y votre pâte sur une épaisseur de 5 mm, piquez-la de quelques coups de fourchette et découpez ce qui dépasse du moule. Disposez harmonieusement des carrés de saumon fumé au fond comme vous le feriez avec des lardons si vous faisiez une quiche normale. Mettez 10 minutes dans le four préchauffé (thermostat 7)*. Pendant ce temps, cassez les 3 œufs dans un bol, notez bien ce mouvement, il constitue la base d'une autre fameuse recette: l'omelette. Battez ces œufs avec la crème, salez à peine, poivrez et rapez un peu de muscade. Versez le mélange sur la pâte à demi-cuite. Ajoutez votre zeste de citron que vous aurez émincé ou rapé. Parsemez de quelques noisettes de beurre. Baissez le four thermostat 5 et faites cuire sans bouillir. Quand la couleur est belle, plantez une lame de couteau (délicatement), si elle ressort propre, vous avez réussi votre recette. C'est bientôt prêt. Sortez la chose. Décorez-la de quelques branches d'aneth. Présentez-la tiède accompagnée d'une petite salade de mâche agrémentée de carrés de betterave, relevée d'une pointe d'ail dans laquelle vous éparpillerez les morceaux d'un petit fromage de chèvre bien sec. Pour la vinaigrette, choisissez

la simplicité: une cuillerée de vinaigre de vin pour trois d'huile. Sel. Poivre et ce que vous trouvez comme herbes sur le rebord de votre fenêtre en privilégiant toutefois persil et ciboulette. Il existe, pour les non jardiniers, des herbes fraîches à acheter chez le marchand.

Pour accompagner ce délicieux petit dîner, sortez de votre cave un Sancerre blanc ou rouge à servir frais (ou faites-le vous offrir par vos amis, après tout, ils vous doivent bien ça!). Ainsi, vos convives se régaleront en poussant ces drôles de petits goussements, signe de leur contentement. ■

* Petite note à l'attention de Carole et tous ceux qui ne comprennent rien à cette affaire de thermostat.

Le thermostat, c'est facile. On est en facteur 30, c'est-à-dire que chaque graduation du thermostat représente 30 °C. Thermostat 5 = 150 °C.

Utile à tous

52 omelettes du dimanche soir,
par Jean-Luc Petitrenaud,
Éditions Europe 1/Minerva, 157 F.

Sur ordonnance du médecin traiteur, on peut même les déguster en semaine.

www.lamartiniere.fr

