Méthodes | Architectures | Technologies | Algorithmie | Langages | Outils | Opinions

DÉVELOPPEUR RÉFÉRENCE

LA LETTRE BIMENSUELLE DU DÉVELOPPEMENT

www.devreference.net

v2.16 | 15 juillet 2002

EDITO

Pour mettre d'accord une bonne fois pour toutes les enragés de Java et les fanatiques de .NET, l'OMG déclare terminée la guerre des middlewares grâce à MDA, l'architecture dirigée par les modèles. Après la POO, la POA, les composants, le modèle de service (services Web)... la prochaine grande évolution du génie logiciel s'appuiera sur les modèles, les méta-modèles et les méta-méta-modèles. Pour comprendre l'enjeu de ce grand chantier, voici le premier volet d'une série d'articles sur MDA, histoire de prendre de bonnes résolutions pour la rentrée.

Avec ce numéro spécial d'été, vous pourrez également mettre en pratique la Programmation Orientée Aspect, surfer sur le Web à la recherche d'un SGBD, ou encore découvrir le versionning sous .NET.

Après la pause estivale, je vous donne rendez-vous fin août pour le numéro de rentrée. Mais comme chacun sait que les vrais développeurs ne prennent pas de vacances... alors bonne lecture!

Pierre Tran
Rédacteur en chef

LE MDA (MODEL DRIVEN ARCHITECTURE) EST UN GRAND CHANTIER QUI S'OUVRE. PLUS DE TRENTE ANS APRÈS LA PREMIÈRE CRISE DU LOGICIEL QUI A EU LIEU DANS LES ANNÉES 68, C'EST UNE NOUVELLE CRISE QUI SECOUE AUJOURD'HUI LE MONDE DE LA PRODUCTION DE LOGICIEL.

La complexité grandissante des systèmes informatiques, leur nécessaire évolutivité et leur couplage de plus en plus fort sur les organisations dans lesquelles ils opèrent ont provoqué une réaction importante et rapide. Pour résoudre les problèmes actuels de construction et d'évolution de systèmes d'information auxquels font face les industriels, l'OMG (Object Management Group) a défini sa nouvelle orientation : MDA (Model Driven Architecture). Le principe de base du MDA est l'élaboration de modèles indépendants des plates-formes (PIM) et la transformation de ceux-ci en modèles dépendants des plates-formes (PSM). Les techniques employées sont donc principalement des techniques de modélisation et des techniques de transformation de modèles. L'OMG a d'ailleurs d'ores et déjà commencé à définir des

[suite page 7]

Sommaire

DÉVELOPPEUR RÉFÉRENCE

LETTRE BIMENSUELLE ÉDITÉE PAR
IDG COMMUNICATIONS FRANCE
5, rue Chantecoq
92808 Puteaux Cedex
Tél. : 01 41 97 61 61

Adresse électronique :
devreference@idg.fr

Directeur de la publication : Ted Bloom
Commission paritaire : en cours
Dépôt légal : 4e trimestre 2001

Rédaction

Editeur
Michel Crestin

Rédacteur en chef
Pierre Tran • tran@idg.fr • 6257

Assistantes de rédaction
Kateline Renaudin

Ont collaboré à ce numéro
Jean-Louis Bénard, Jean Bézivin, Xavier Blanc, Pierre Chalamet, Ramnivas Laddad, Guillaume Louel, Frédéric Milliot

Fabrication

Directeur artistique groupe
Patrice Servage

Mise en page
Pierre Tran

Photographies
Marc Guillaumot

Iconographie
Nouara Aftis

Infographies
Pierre Tran

Publicité

Pasquale Meyer • meyer@idg.fr • 6216
Caroline Mayer • mayer@idg.fr • 6217

Service Abonnements
Développeur Référence
BP 90006 – 59718 Lille Cedex 9
Tél. 03 20 12 11 17
Fax 03 20 12 86 09

Tarif :
1 an 22 numéros : 200 euros
Prix de lancement : 150 euros

Renseignements :
Lucienne Bosser • bosser@idg.fr • 6128

Evénement

EuroPython 2002

Du 26 au 28 juillet se déroulait à Charleroi en Belgique EuroPython 2002, la première conférence européenne consacrée aux technologies Python et Zope.

Par Guillaume Louel

Zope est devenu depuis quelques temps le sujet de nombreuses attentions. A mi chemin entre un serveur d'applications et un système de gestion de contenu, il permet de réaliser facilement des portails de contenu, mais aussi toutes sortes d'applications. Zope est programmé, et se programme en Python. C'est l'application phare de ce langage, et c'est d'ailleurs tout naturellement que l'auteur de Python, Guido van Rossum est employé par Zope Corp. Cette première conférence européenne était l'occasion de réunir de nombreux développeurs Zope venant d'un peu partout en Europe, certains venant même de Russie. Cette conférence accueillait de nombreux acteurs du monde du logiciel libre d'Europe ou d'ailleurs, comme Eric S. Raymond, mais également de la communauté Python/Zope, y compris Guido van Rossum, l'auteur de Python, et Jim Fulton, le programmeur principal de Zope. Outre les traditionnelles et non moins intéressantes sessions consacrées à l'avenir des différentes technologies (la présentation de Zope 3 par Jim Fulton était remarquable), on a retrouvé des sessions plus pratiques consacrées au développement en Python, sur l'utilisation de XML, la gestion des threads, ou encore la création d'extensions en C.

Un certain nombre de sessions étaient consacrées à des applications réalisées sous Zope, comme le très impressionnant Silva, un système d'authoring et de publication en ligne à grande échelle réalisé pour une université, chaleureusement accueilli par l'audience présente. Ce fut également l'occasion de présenter Plone, l'extension de CMF qui, selon les dires mêmes de Paul Everitt — l'évangéliste de Zope — "va nous sauver de nous-mêmes". Cette extension transforme CMF en un véritable système de gestion de contenu, certains n'hésitant pas à le désigner comme le PostNuke de Zope.

La dernière journée de conférence était consacrée en partie à la présentation de cas clients, comme l'utilisation de Zope dans l'e-commerce, ou l'utilisation de Python dans l'astronomie. La matinée s'est conclue sur des sessions désormais classiques dans les conférences liées au monde de l'Open Source, consacrées à la manière de réaliser des profits avec l'Open Source, ou de présenter l'Open Source à de grands comptes.

Avec près de 200 participants, cette conférence était également l'occasion pour tout ceux qui étaient présents de rencontrer les différents acteurs de la communauté Zope, et de discuter concrètement des problèmes et des améliorations à apporter aux différents composants et applications. Une chance unique pour les développeurs, qui ont pu discuter lors de tables rondes des implémentations actuelles, et décider ensemble des voies à suivre pour les prochaines versions.

Interview de Paul Everitt

Nous avons rencontré le co-fondateur et évangeliste de Zope, qui évoque le statut actuel de Zope et son avenir.

Développeur Référence : Pouvez vous nous décrire votre rôle dans le monde de Zope ?

Paul Everitt : Je suis le co-fondateur de Zope Corporation, et j'en suis actuellement le Chef Stategy Officer. Concrètement, je joue un rôle d'évangeliste dans la communauté Zope, mais ce que j'apprécie le plus dans cette activité, c'est de se faire rencontrer des gens, afin qu'ils puissent construire de nouvelles relations de travail.

Si vous deviez définit Zope en quelques mots ?

Zope est un serveur d'application Open Source, qui se focalise sur la gestion de contenu, et la création d'applications customisées pour Internet. Pour les développeurs, Zope est surtout un framework, permettant la publication sur Internet d'objets Python.

Python n'est pas un langage extrêmement populaire, quelles sont les motivations qui vous ont poussés à le choisir pour la réalisation de Zope ?

Tout d'abord, il faut savoir que nous sommes issus Jim [ndlr : Jim Fulton, le programmeur principal de Zope] et moi de la communauté Python. A l'époque, en

94-95, Java n'était pas encore très populaire, et nous avons commencer à créer une architecture de publication d'articles, qui à été dans un premier temps un logiciel commercial. C'est de ce logiciel qu'a dérivé Zope, qui est désormais une application Open Source. Le choix de Python que nous avions fait par rapport a Java n'est pas si mauvais, car que je sache, Java n'a pas encore raflé tout le marché : c'est donc qu'il y a besoin d'une alternative. Python est avant tout un langage très puissant, et très simple a apprendre. La plupart des applications Web en ce moment fonctionnent encore sur le bon vieux principe des CGI. Grâce à Python, nous avons fait de Zope un système qui permet de créer sans problèmes de gros systèmes (bien plus facilement qu'avec des CGI), mais aussi de réaliser rapidement des applications plus simples.

La documentation est souvent l'un des points faibles de l'Open Source, il semble que Zope ne soit pas épargné par ce problème ?

Il faut bien distinguer deux types différents de documentation. La documentation de base, le Zope Book, est en très bonne forme, il est régulièrement maintenu et couvre les généralitées de Zope. Cependant, Zope étant une architecture objet, de nombreuses classes ont été et sont encore ajoutées au fur et à mesure, et c'est généralement de ce coté que la documentation fait défaut. Nous avions laissé cette tâche de documentation annexe à la communauté d'utilisateurs et de développeurs, qui ont fait un travail formidable, mais il est vrai que nous n'avons peut-être pas assez tenté de fédérer les efforts, ou de tenter de maintenir à jour les différentes documentations disponibles, certaines étant malheureusement obsolètes. Nous allons consacrer beaucoup de temps dans les mois à venir sur la documentation, car c'est une étape indispensable si l'on souhaite que Zope évolue et prenne une place de plus importante sur le marché. C'est une étape à franchir pour atteindre la maturité.

Que diriez-vous à un développeur PHP ou CGI pour le convaincre de passer sous Zope ?

Paul Everitt, co-fondateur de Zope : "Notre objectif : multiplier par dix en un an, en grande partie grâce à Zope 3."

Tout d'abord, il faut bien mettre les choses au point. Si vous êtes un développeur isolé qui réalise l'application de A à Z seul dans son coin, restez avec PHP, vous ne profiterez pas réellement des avantages de Zope. Si par contre vous travaillez en équipe, les choses sont toutes autres, vous profiterez réellement de l'aspect collaboratif, et de la vision de haut niveau qui permet de séparer très clairement les tâches. L'avantage de Zope est qu'il vous permet de travailler à un plus haut niveau que les langages traditionnels.

Mais le meilleur conseil que je puisse donner est de regarder les applications existantes. Vous souhaitez voir quelque chose qui marche tout de suite, essayez Plone, une sorte de PostNuke pour Zope. Et n'hésitez pas à venir demander de l'aide a la communauté, que ce soit par IRC, sur les mailing lists, ou sur des événements comme ici avec EuroPython.

Vous n'hésitez pas a remettre en cause en permanence l'existant, de plus en plus, vous délaissez le DTML au profit d'un nouveau modèle de templates ?

C'est exact. Le DTML était un langage de templates un peu particulier, dans le sens ou il disposait d'un certain nombres de macros, des fonctionnalités de haut niveau rajoutées à HTML qui permettaient de créer certaines tâches. Le problème est que ce genre de langage est totalement anti-productif dans un cadre de développement collaboratif, les Web designers qui s'occupent de l'aspect graphique suppriment très souvent tout ou partie de votre code, le rendant totalement non fonctionnel. C'est pour cela que nous avons introduit les Zope Page Templates (ZPT), qui répondent un peu mieux à ce besoin. Seul le code réellement indispensable à l'affichage est placé dans les templates, et la grande force est que nous plaçons ce code en tant que paramètres des balises HTML existantes. Cela donne par exemple pour un paragraphe quelque chose du genre

```
<p tcl :content= "here/title">
```

beaucoup plus simple à programmer, beaucoup plus clair, et cela facilite grandement le travail collaboratif.

Pouvez vous nous résumer les principales directions pour le développement de Zope 3 ?

Nous cherchons à atteindre la maturité avec la prochaine version de Zope. Pour l'instant, Zope remplit très bien sa tâche de CMS (Content Management Système, Système de gestion de contenu), mais il nous reste un peu de travail dans le domaine de la création d'applications d'autres types. L'architecture a été complètement revue, sous forme de composants afin qu'elle soit plus facile à maintenir, mais aussi plus facile à faire évoluer car plus modulaire.

Nous essayons aussi de faire un gros travail afin de faciliter l'apprentissage de Zope, nombreux sont les développeurs qui se sentent découragés lorsqu'ils se penchent pour la première fois sur Zope, entre autre parce que nous utilisons des concepts radicalement différents d'objets, mettant de coté le paradigme des pages. Après maintes discussions, nous avons décidés de réintroduire une notion de page dans Zope 3, le but n'est pas de revenir en arrière, mais juste d'offrir un choix supplémentaire aux développeurs afin de faciliter la phase d'apprentissage. C'est très important si nous souhaitons réaliser notre objectif de progression.

Et quel est votre objectif ?

Multiplier par dix, en un an. En grande partie grâce à Zope 3.

C'est une prévision qui est peut être un peu optimiste, quels sont selon vous les clés qui vous permettront de réaliser cette progression, par rapport aux autres systèmes présents sur le marché ?

Il faut tout d'abord prendre en compte les coûts, et pas seulement les coûts logiciels. Au niveau des coûts matériels, et des coûts de maintenance, l'Open Source offre également des coûts extrêmement compétitifs. Il faut voir également quels sont nos concurrents. A votre avis quel est le leader du marché des CMS ? Vignette ? Pas du tout, avec 43% des parts de marché, ce sont les systèmes créés en interne dans les entreprises qui jouent le rôle de CMS. Des systèmes généralement réalisés à la va-vite, parce qu'il fallait être présent sur Internet.

Désormais, nous entrons dans une phase de renouvellement, où l'on s'aperçoit que ces systèmes sont difficiles à faire évoluer et ne répondent plus aux besoins, que ce soit en termes de fonctionnalités ou de sécurité. Des problématiques auxquelles un CMS Open Source comme Zope répond parfaitement.

Gérer la connaissance (2)

Le repérage

Jean-Louis Bénard
est directeur technique de Business Interactif, qui met en place des solutions e-business (front, middle et back-office) sur des architectures multitiers à base d'objets métier.
jlb@businessinteractif.fr

Nous avons vu dans le précédent numéro qu'un projet de gestion des connaissances obéissait à un cycle : repérer, préserver, valoriser et actualiser les connaissances. Dans ce deuxième volet du dossier, nous allons nous intéresser au "repérage", c'est-à-dire à l'identification des connaissances cruciales au sein des équipes informatiques.

Il est assez amusant de constater à quel point la notion de connaissance cruciale est subjective : lorsqu'on demande à un développeur junior ce qui lui semble essentiel, les réponses sont souvent très différentes d'un développeur senior. L'un attachera une importance toute particulière aux technologies du moment, l'autre à des schémas de conception (design patterns) indépendants du langage ou du serveur d'application mis en œuvre.

L'identification des connaissances essentielles est donc une étape particulièrement complexe. Elle dépend à la fois du profil des individus (développeur, chef de projet, architecte, ingénieur réseaux), mais aussi de l'entreprise dans laquelle évolue les ces individus (entreprise utilisatrice, société de services ou de conseil, éditeur de progiciels...). La cartographie que nous présentons ici est donc à adapter en fonction d'un certain nombre de paramètres, et ne saurait constituer à elle seule une cartographie exhaustive.

De manière globale, il est possible de regrouper les connaissances en deux groupes : les connaissances technologiques et les connaissances méthodologiques. Le focus est souvent mis — à tort — sur les connaissances technologiques alors qu'elles masquent des besoins beaucoup plus forts. Bien souvent la maîtrise de connaissances méthodologiques permet de décupler la vitesse d'apprentissage de connaissances technologiques (identification de design patterns réapplicables dans des environnements technologiques différents). Pour chacun des profils type, à commencer par celui du développeur, nous allons maintenant esquisser les éléments permettant d'effectuer une cartographie de ces connaissances technologiques et méthodologiques.

Les connaissances technologiques

Les connaissances technologiques sont instinctivement celles qui émergent le plus spontanément : connaissance des langages (Java, C#, SQL,…), des environnements de développement, des frameworks (J2EE ou .NET)... Ces connaissances sont complexes à cartographier, et ce pour plusieurs raisons :

• Une grande partie sont des connaissances "volatiles", à durée de vie limitée : elles passent au rythme des technologies imposées par les éditeurs. Si la connaissance d'OLE ou plus récemment de COM était fondamentale, elle le sera probablement beaucoup moins dans deux ans.

• D'autre part, pour chacune de ces connaissances technologiques, il est essentiel de ne pas assimiler le torrent d'information déversé par les éditeurs aux connaissances cruciales du développeur : les connaissances fondamentales sont en fait celles qui permettent d'aboutir à la mise en œuvre efficace des technologies dans le cadre de l'entreprise. Elles résultent d'une savante alchimie entre les informations technologiques fournies par l'éditeur, le cadre d'utilisation défini par l'entreprise et le "savoir-faire", beaucoup plus complexe à formaliser. Un exemple simple illustre ceci : un développeur "certifié" n'est pas forcément un bon développeur. La maîtrise d'informations bachotées ne garantit en rien le savoir-faire, le "how-to" qui distingue un bon développeur d'un mauvais. Dans le même esprit, on s'intéressera également aux éléments qui permettent au développeur de passer d'un stade de connaissance à un autre : à quoi doit-on le fait qu'un développeur junior devient senior ? Ce ne sont donc pas seulement les connaissances d'expertise qui doivent être adressées, mais également celles devant être assimilées à chacune des étapes d'apprentissage du collaborateur.

Le travail de cartographie des connaissances technologiques indispensables consistera donc à discerner les éléments technologiques récurrents, qui ne "passent pas de mode" de ceux dont la volatilité est très forte. Ce premier axe de classement est essentiel car les moyens de diffusion et de capitalisation des connaissances devront être adaptés à leur

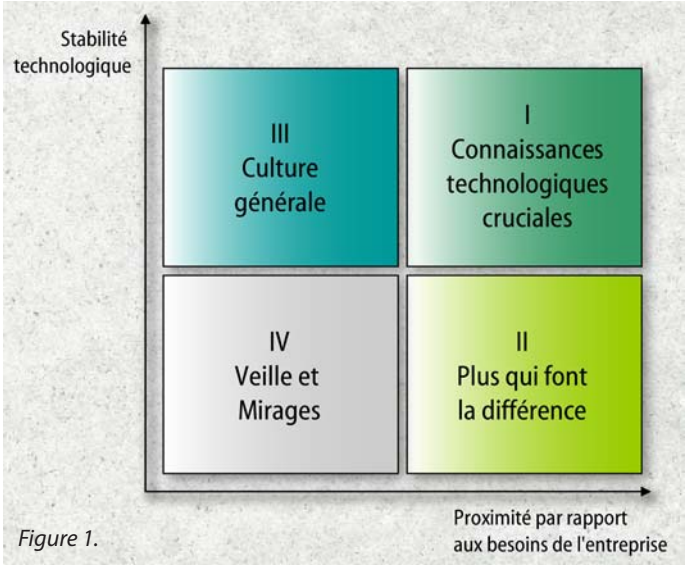


Figure 1.

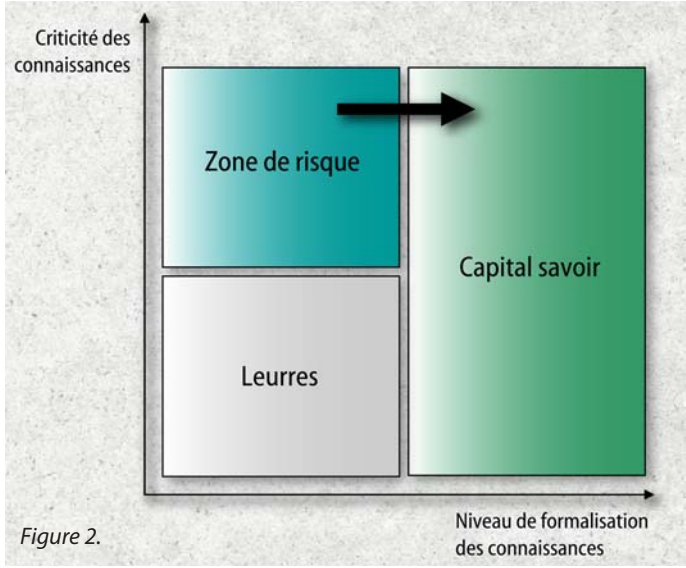


Figure 2.

volatilité, comme nous le verrons dans les articles suivants. Le deuxième axe de classement consiste à positionner les connaissances selon leur degré de pertinence par rapport au contexte : pour une entreprise qui réalise des applications client-serveur en environnement transactionnel, la cartographie des connaissances essentielles à la mise en œuvre de .NET n'aura pas grand-chose à voir avec une entreprise réalisant des applications de consultation de données en environnement Web, même si quelques fondamentaux constitueront un dénominateur commun. Les connaissances peuvent donc être positionnées sur la matrice de la figure 1.

Le cadran I correspond aux connaissances technologiques essentielles, indispensables et stables. On pourra par exemple ranger dans ce cadran la connaissance des langages de développement mis en œuvre par l'entreprise (Java, VB…), la connaissance des environnements de développement pratiqués, etc.

Le cadran II matérialise les éléments technologiques qui impactent directement l'entreprise, mais qui ont un niveau de volatilité très fort. Ce peut être par exemple la mise en œuvre de COM, puis de COM+, etc. Ces éléments évoluent à des rythmes technologiques très soutenus, imposés par les éditeurs. C'est souvent la capacité de l'entreprise à s'adapter rapidement à ces changements et à tirer partie de la valeur ajoutée associée qui crée l'avantage concurrentiel : ce sont les "plus qui font la différence". Les processus et les outils associés à l'apprentissage de ces connaissances volatiles devront donc être adaptés en conséquence.

Le cadran III correspond aux connaissances stables mais qui ne sont pas dans le périmètre d'intérêt actuel de l'entreprise. Ces connaissances non immédiatement applicables doivent néanmoins faire l'objet d'attentions, parce qu'elles permettent à l'entreprise de mieux faire face à des changements d'environnement, d'objectifs business qui influent sur le système d'information.

Le cadre IV regroupe les éléments qui constituent la veille technologique pure : connaissances à forte volatilité qui ne rentrent pas directement dans le périmètre d'intérêt immédiat de l'entreprise. C'est aussi le cadre des "mirages" : connaissances dont l'entreprise pense un jour avoir besoin, mais qui en réalité ne sont et ne seront jamais créatrices de valeur pour elle. La mise en œuvre de la veille constitue là aussi un exercice complexe pour assurer son adéquation aux besoins futurs réels de l'entreprise.

Les connaissances méthodologiques

Les connaissances méthodologiques du développeur sont quant à elle très stables. Elles sont un support à l'ensemble de ses activités et ne dépendent que peu des cycles technologiques. Ce sont en particulier les connaissances liées à la modélisation de données et d'objets, à l'algorithmie, aux design patterns et au méthodes de tests. La formalisation des design patterns (au-delà de ceux préconisés par le " gang of four ") constitue naturellement une étape clé dans la capitalisation des connaissances, étape sur laquelle nous aurons l'occasion de revenir. Ces connaissances méthodologiques doivent souvent être complétées par un apprentissage de la formalisation orale et écrite (un vrai problème) : rédaction de documents, expression orale, qui laissent souvent à redire.

Prioriser les connaissances en fonction de contexte

Ce travail de cartographie doit être complété par un certain nombre d'éléments, qui permettent d'identifier les connaissances les plus cruciales pour l'entreprise. Il est essentiel de hiérarchiser les connaissances cartographiées. En effet, selon le département informatique considéré, les priorités ne seront pas nécessairement les mêmes : les connaissances cruciales d'équipe de développement de nouveaux produits ne seront ainsi pas les mêmes que les connaissances d'équipes de maintenance applicative, même si elles travaillent dans le même environnement technologique. Enfin pour chaque contexte, on essaiera d'identifier les connaissances les plus vitales. Ce travail peut se faire notamment en imaginant le départ de chaque collaborateur : quelles sont les connaissances perdues ? Comment la perte de ces connaissances pénalise-t-elle l'activité de l'entreprise ? Combien de temps faudrait-il pour acquérir à nouveau ces connaissances ? Autant de questions qui aideront à positionner des niveaux de criticité par rapport aux connaissances cartographiées. Ces niveaux permettront de mettre en œuvre une politique de préservation ciblée, sur laquelle nous reviendrons plus tard.

Attribuer un niveau de formalisation

Enfin pour chacune des connaissances cartographiées, on s'attachera à identifier le niveau de formalisation actuel : les connaissances se transmettent-elles simplement par voie orale, de manière informelle, ou au contraire font-elles l'objet de documents de référence réellement exploités par les collaborateurs.

Les connaissances peuvent donc être positionnées sur la matrice de la figure 2. Les travaux devront adresser en priorité le cadran "zone de risque", connaissances critiques mais tacites, pour les rendre explicites.

Dans le prochain article, nous nous intéresserons aux connaissances technologiques et méthodologiques associés aux profils de chef de projet, d'architecte et d'ingénieur réseaux. Nous verrons ensuite quels sont les processus et les outils à mettre en place pour préserver ces connaissances.

Palais des Congrès de Versailles

3 et 4 Octobre 2002

Le rendez-vous de la Technologie JAVA

Un événement co-produit par

Scope 2002

Echangez, Partagez autour de la Technologie JAVA

Inscriptions dès le 15 Juillet

JAVA, J2EE, J2ME, J2SE, EAI, XML,
WSAD, Eclipse, JBuilder, 9iAS, Sun ONE...
construisent notre avenir

- 2 Séances plénières stratégiques
- 1 Cycle de 40 Ateliers thématiques
  - Java et les webservices
  - Java dans les mobiles
  - IHM en Java
  - Le développement rapide en Java
- Des Testimoniaux majeurs exprimant les retours d'expériences
- Une Exposition résolument orientée technique

www.java-scope2002.com

Jean Bézivin
est professeur d'informatique à l'Université de Nantes. Il a participé activement dans les années 1980 à la mise en place de la communauté objet en France et en Europe, en créant avec P. Cointe la série de conférences ECOOP, avec B. Meyer la série de conférences TOOLS, avec S. Caussarieu et Y. Gallison la série de conférences Objet/OCM et plus récemment avec P.-A. Muller la série de conférences UML. Il est l'auteur de nombreuses publications de recherche et de tutoriels dans des conférences internationales. Il est animateur du groupe de travail de l'OFTA (Observatoire Français des Techniques Avancées) sur l'ingénierie de la modélisation. Jean.Bezivin@Sciences.Univ-Nantes.fr

Xavier Blanc
est chef de projets R&D à SOFTEAM et docteur de l'Université Paris 6. Ses travaux ont porté sur les techniques de modélisation, de méta-modélisation et sur les transformations de modèles en utilisant le standard XMI. Il a été membre du groupe ISO qui a défini le point de vue entreprise de la norme RM-ODP. Il a participé à la création de l'ORB JavaORB plus connu aujourd'hui sous le nom de OpenORB. Il est actuellement le rapporteur de l'OFTA (Observatoire Français des Techniques Avancées) sur l'ingénierie de la modélisation. Xavier.Blanc@Softeam.fr

[suite de la page 1]

standards dans ce domaine. Les plus connus sont bien sûr UML et XMI mais les autres tels que MOF, CWM ou les profils EJB et Corba ne sont pas moins indispensables. D'autres sont à venir, par exemple sur les transformations de modèles ou sur les services web. L'idée est donc séduisante, mais sa mise en œuvre n'en est encore qu'à ses débuts. Il faudra en effet surmonter des défis techniques comme la modélisation des transformations de modèles, mais aussi des défis méthodologiques et stratégiques pour que l'approche MDA puisse s'implanter dans les entreprises.

Le présent article a pour but de situer l'initiative MDA dans ses caractéristiques générales. Il est le premier d'une série qui vise à présenter plus en détail la nouvelle démarche basée sur les modèles préconisée par l'OMG ainsi qu'à en suivre les évolutions.

Une montée en abstraction nécessaire

En novembre 2000 à Orlando, l'OMG a rendu publique sa nouvelle orientation dans un document intitulé MDA (Model Driven Architecture) [1]. Il s'agit du constat, plus de dix ans après la création de l'OMG, que la recherche de l'interopérabilité dans les systèmes d'information ne peut être atteinte uniquement grâce à la standardisation des infrastructures de middleware, mais par une approche beaucoup plus globale où la notion de modèle devient essentielle et centrale.

En schématisant, on pourrait dire que l'OMG déclare terminée la guerre des middlewares. Chacun est installé dans son territoire et il serait stérile de tenter d'imposer un ultime dernier middleware. Cette analyse de la situation n'est pas nouvelle, l'abandon du "tout-Corba" avait d'ailleurs été déjà amorcé depuis longtemps avec la fourniture de ponts vers d'autres plates-formes. Mais, il faut se rendre à l'évidence, la construction de multiples ponts entre middlewares ne peut constituer ni un objectif réaliste de long terme ni la base d'une stratégie [3].

Aujourd'hui la nouvelle stratégie MDA se démarque très nettement de l'ancienne (OMA : Object Management Architecture), en se positionnant clairement au niveau supérieur d'abstraction et en mettant le projecteur non plus sur les approches à objets mais sur les approches à modèles [4]. Le but est de se concentrer sur l'élaboration de modèles de plus haut niveau d'abstraction et de favoriser les approches transformationnelles paramétriques vers les plates-formes techniques. Ceci signifie que l'on pourra, par exemple, définir des modèles métier totalement indépendants des plates-formes techniques et générer du code soit pour Corba, soit pour Java/EJB, soit pour C#/.NET, soit pour XML/Soap/WSDL, soit encore pour des compositions de ces différentes plates-formes.

L'initiative MDA repose sur un changement de paradigme important dans le domaine du génie logiciel. La rapidité de cette évolution s'explique par au moins trois facteurs.

Le premier facteur est la nécessité de découpler les éléments que l'on peut considérer comme stables dans les logiques métier, de l'évolution des supports technologiques informatiques. Cette évolution dont on a finalement accepté le caractère non conjoncturel, inéluctable et permanent ne va pas se ralentir. Cependant, il ne faut pas que cette évolution, si profitable qu'elle soit, pénalise la capitalisation logicielle du savoir et du savoir-faire de l'entreprise. L'objectif est donc clair,

même si les stratégies pour l'atteindre ne sont pas encore complètement explicitées.

Deuxième facteur, la relative incapacité de la technologie des objets et des composants à tenir les promesses faites il y a une vingtaine d'années. La complexité croissante des déploiements logiciels basés sur les formes modernes de composants comme les EJB est contraire aux espoirs de simplification conceptuelle des premiers évangélistes de l'objet. Le recours régulier à des expédients technologiques (cas d'utilisation, patterns de conception, programmation par aspects, etc.) ne fait que renforcer le sentiment que la technologie des objets n'est pas le remède miracle du génie logiciel (le "silver bullet" comme disent nos collègues américains) que l'on nous a parfois fait miroiter [2].

Enfin troisième facteur, l'urgence de stopper l'empilement des technologies. Nos systèmes ressemblent parfois à des châteaux de cartes construits à partir d'une mosaïque hétéroclite de technologies, pour lesquelles il est néanmoins nécessaire de conserver dans l'entreprise des compétences diverses et variées pour des raisons de maintenance corrective ou préventive. Cette situation est de plus en plus mal vécue. Pour permettre au moins une pause et la possibilité d'un point d'arrêt à cette "course aux armements" des sur-couches de logiciels, la parade consiste à remplacer les approches interprétatives par des approches transformationnelles permettant de générer des systèmes cibles à empreintes plus légères, moins gourmands en ressources car plus finement dimensionnés aux besoins réels minimaux des applications.

Un grand chantier s'ouvre

Le MDA est un nouveau chantier qui s'ouvre et on peut raisonnablement penser qu'il va durer au moins autant que le précédent chantier Corba (c'est-à-dire plus de douze années). Les premiers travaux déjà visibles illustrent pleinement les avantages de cette stratégie. Parmi ceux-ci, nous pouvons citer, entre autres, UML qui est le standard OMG de modélisation, XMI qui est le standard OMG de sérialisation de modèles basé sur XML, le profil UML EDOC pour les modèles d'entreprise et les profils UML EJB et Corba. Grâce à ces standards, il est d'ores et déjà possible d'envisager la transformation automatique de modèles métier indépendants de plates-formes techniques vers des modèles dépendants de ces plates-formes EJB et Corba.

Par ailleurs, l'un des grands débats actuels est celui de l'émergence des modèles dits de "service" (services Web, par exemple) par rapport aux modèles classiques d'objets et de composants. Les modèles de service font leur retour sur le devant de la scène pour plusieurs raisons. Ils se combinent beaucoup mieux que les modèles d'objets avec les modèles de workflow et de processus métier. Ils permettent d'intégrer beaucoup plus facilement les aspects fonctionnels et non fonctionnels (qualité de service, etc.). Ils sont déployables de façon simple vers des contextes Web ou systèmes transactionnels. Enfin ils avaient été les oubliés lors du passage de la technologie procédurale à la technologie des objets (les "cas d'utilisation" ne prenant en compte qu'une partie très limitée des besoins). L'intérêt de l'ingénierie de la modélisation est de pouvoir concilier ces différentes tendances et de faciliter, par exemple, le passage de la spécification par objets à la spécification par services.

De façon réaliste, les auteurs de la spécification [1] restent lucides sur la maîtrise actuelle du processus de transformation de modèles. Ils envisagent une approche semi-automatique ou assistée dans un premier temps. Par la suite, il n'est pas exclu de voir s'imposer des techniques partielles de transformation. Le travail à faire sur l'industrialisation des techniques automatiques ou assistées de transformation de modèles reste considérable. Il est à noter qu'un recensement en cours des outils actuellement disponibles (industriellement et commercialement) donne des indications très positives (et même surprenantes) sur la rapidité d'adaptation du tissu industriel à la nouvelle donne de l'ingénierie des modèles.

Le principe de base : des PIM vers les PSM

L'approche MDA suscite actuellement un très grand intérêt. Elle concrétise une évolution très marquée de la technologie informatique visant à substituer aux approches interprétatives (adjonctions de couches logicielles pour prendre en charge les problèmes d'interopérabilité) des approches transformationnelles (traduction des logiques métier neutres vers des modèles liés aux plate-formes d'exécution).

D'une façon macroscopique, le MDA peut se résumer à l'élaboration de modèles indépendants de plates-formes et à la transformation de ceux-ci en modèles dépendants de plates-formes. Plus précisément, l'approche MDA définit deux types de modèles :

- Les **PIM** (Platform Independant Models) sont des modèles qui n'ont pas de dépendance avec les plates-formes techniques (c'est-à-dire EJB, Corba, .NET, XML, etc.). Les PIM représentent par exemple les différentes entités fonctionnelles d'un système avec leurs interactions, exprimées uniquement en termes de logique d'entreprise.
- Les **PSM** (Platform Specific Models) quant à eux sont dépendants des plates-formes techniques. Les PSM servent essentiellement de base à la génération de code exécutable vers ces mêmes plates-formes techniques.

Le principe de l'approche MDA est bien sur de passer des PIM aux PIM pour permettre facilement la génération de code vers une plate-forme technique choisie. Ce passage des PIM aux PSM est une transformation de modèles. Le MDA identifie plusieurs types de transformations entre les PIM et les PSM :

1. PIM vers PIM : Ces transformations s'effectuent pour ajouter ou soustraire des informations aux modèles. Le fait de masquer par exemple quelques éléments afin de s'abstraire de détails fonctionnels est typiquement une transformation PIM vers PIM. Dans l'autre sens, le passage de la phase d'analyse à la phase de conception est la plus naturelle des transformations PIM vers PIM. Il est important de

ALLER PLUS LOIN

[1] Model-Driven Architecture
R. Soley and the OMG staff, novembre 2000. OMG document disponible sur : http://www.omg.org

[2] From Object Composition to Model Transformation with the MDA
TOOLS'USA 2001, J. Bézivin, Santa Barbara, August 2001, Published in Volume TOOLS'39, IEEE publications, Los Alamitos, California, ISBN 0-7695-1251-8, pp. 346-354.

[3] Model-driven approaches to software development
S. Cook, présentation invitée aux journées OCM 2002, Ecole des Mines de Nantes, France, mars 2002.

[4] A Metamodel-Based Approach to Model Engineering Using the MOF
R. Dirckze, S. Iyengar, J. Bézivin and J. Ernst, eds, First International Workshop on Model Engineering, Nice, France, (June 13, 2000), article disponible sur : http://www.metamodel.com

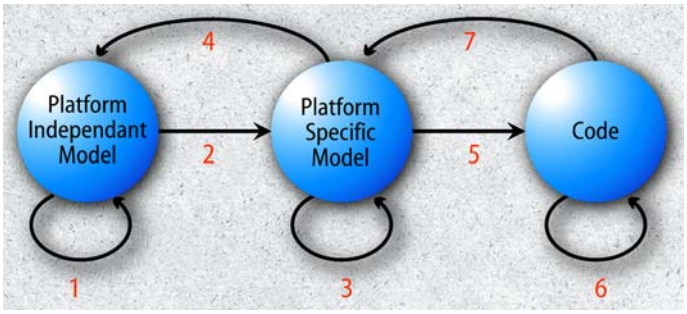


Figure 1. Les différentes opérations sur les modèles dans le MDA.

noter que ces transformations ne sont pas toujours automatisables.

2. PIM vers PSM : Ces transformations s'effectuent lorsque les PIM sont suffisamment complets pour pouvoir être "immergés" dans une plate-forme technique. L'opération qui consiste à ajouter des informations propres à une plate-forme technique pour permettre la génération de code est une transformation PIM vers PSM. A l'heure actuelle, les plates-formes techniques visées sont .NET, J2EE, XML et Corba. Il apparaît clairement que ce sont les règles qui permettent ces transformations qui sont importantes et qui doivent être généralisées et capitalisées. Ces transformations ont donc pour but d'être fortement automatisées.

3. PSM vers PSM : Ces transformations s'effectuent lors des phases de déploiement, d'optimisation ou de reconfiguration. Notons de plus qu'une unique transformation PIM vers PSM n'est pas toujours suffisante pour permettre la génération de code, il faudra alors parfois transformer les PSM en PSM en utilisant des formalismes intermédiaires (exemple de passage d'UML à SDL puis de SDL à C++).

4. PSM vers PIM : Ces transformations s'effectuent pour construire des PIM à partir d'un existant. Même quand le patrimoine applicatif peut être représenté sous forme de PSM, ces transformations sont assez difficiles à établir dans l'état de l'art actuel. Elles sont pourtant nécessaires à considérer dans toute stratégie de migration.

La figure 1 présente les différentes transformations de modèles identifiées par l'approche MDA. Il est important de noter que la génération de code n'est pas toujours considérée comme une transformation de modèles même si certains font la distinction entre PSM exécutables (le code) et PSM non exécutables.

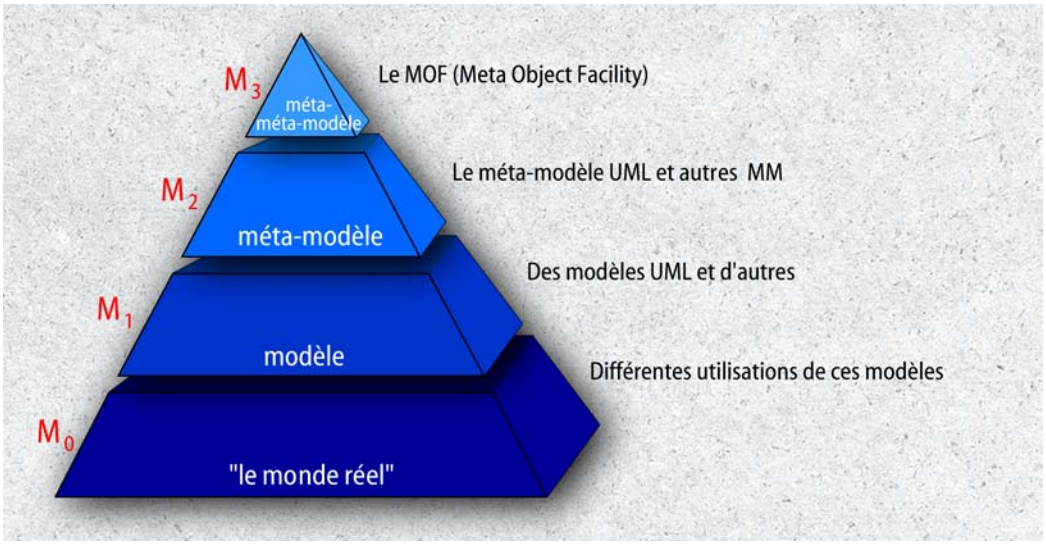
Le besoin d'une architecture stratifiée

La distinction entre les PIM et les PSM est plutôt d'ordre méthodologique. Il est important de noter que cette distinction n'est pas la seule. En effet, les modèles manipulés dans le MDA sont de natures très diverses. Il y a bien sûr les modèles d'objets métier, les modèles de processus ou de workflow, les modèles de règles mais aussi les modèles de service, de qualité de service, les modèles de plate-forme, les modèles de transformation, les modèle de tests, les modèles de mesure, etc. Derrière le recensement en cours de tous ces modèles (la cartographie générale du MDA) se profile un projet ambitieux et à long terme de définition d'un cadre générique de conception et de maintenance des systèmes d'information.

Afin d'organiser et de structurer tous ces modèles, l'OMG a défini une "Architecture à quatre niveaux" (figure 2). Ces quatre niveaux sont :

- Le niveau M0 qui est le niveau des données réelles, composé des informations que l'on souhaite modéliser. Ce niveau est souvent considéré comme étant le monde réel.
- Le niveau M1 est composé de modèles d'informations. Lorsque l'on veut décrire les informations contenues dans le niveau M0, le MDA considère que l'on fait un modèle appartenant au niveau M1. Typiquement, un modèle UML appartient au niveau M1. Tout modèle est exprimé dans un langage unique dont la définition est fournie explicitement au niveau M2.
- Le niveau M2 est donc composé de langages de définition des modèles d'information, appelés aussi méta-modèles. Un méta-modèle définit la structure d'un de modèle. Typiquement, le méta-modèle UML qui est décrit dans le standard UML appartient au niveau M2; il définit la structure interne des modèles UML. Notons au passage que le concept de profil UML qui permet d'étendre le méta-modèle UML est aussi considéré comme appartenant au niveau M2. Un méta-modèle définit donc un langage de modélisation spécifique à un domaine et un profil définit un dialecte (une variante) de ce langage.
- Le niveau M3 qui est composé d'une unique entité qui est le langage unique de définition des méta-modèles, aussi appelé le

Figure 2 L'architecture à 4 niveaux.



méta-méta-modèle ou le MOF (Meta-Object Facility). Le MOF définit la structure de tous les méta-modèles qui se trouvent au niveau M2. Le MOF est réflexif, c'est-à-dire que la description du modèle MOF est faite par le modèle MOF lui-même, ce qui permet de dire que le niveau M3 est le dernier niveau de l'architecture. Le niveau M3 correspond donc aux fonctionnalités universelles de modélisation logicielle, alors que le niveau M2 correspond aux aspects spécifiques des différents domaines, chaque aspect particulier étant pris en compte par un méta-modèle spécifique.

Dans cette architecture, les PIM et les PSM appartiennent au niveau M1. Leurs structures sont définies par des méta-modèles (ou profils) qui appartiennent au niveau M2. Le MOF permet de définir de façon homogène, tous ces méta-modèles. Le MDA ne traite pratiquement pas du niveau M0.

Il faut enfin insister sur le fait que les niveaux qui viennent d'être décrits n'ont rien à voir avec des niveaux d'abstraction ou de machines virtuelles comme on peut en trouver dans les piles ISO/OSI ou TCP/IP par exemple. Ce sont des niveaux de méta-modélisation (également différents de ce que l'on nomme parfois la méta-programmation).

Les standards de l'OMG

L'OMG a déjà défini plusieurs standards pour le MDA : nous en dressons ici une liste des plus importants.

MOF (Meta Object Facility) [6]

Comme nous l'avons déjà vu, le MOF est l'unique entité du niveau M3. Il définit la structure de tous les méta-modèles du niveau M2 comme UML, CWM et SPEM. L'intérêt principal du MOF est surtout qu'il permet la définition d'opérations globales non spécifiques applicables tous les méta-modèles. L'exemple le plus significatif est XMI qui permet l'échange de modèles quel que soit leur méta-modèle. Un autre exemple est la fourniture d'API définies en IDL permettant la manipulation par programmation de modèles.

UML (Unified Modeling Language) [8]

UML est le standard de modélisation des logiciels à objets qu'il n'est plus nécessaire de présenter. UML permet la modélisation d'architectures, de structure d'objets, d'interactions entre ces objets, de données, de composants, de framework, etc. Le MDA préconise fortement l'utilisation d'UML pour l'élaboration de PIM et de PSM.

A l'heure actuelle, UML en est à sa version 1.4 et l'OMG travaille sur la version 2.0 qui devrait voir le jour en 2003. Cette version 2.0 contiendra entre autre une évolution du langage d'expression de contrainte OCL, des concepts renforcés pour les composants et s'intégrera plus facilement dans le MDA grâce notamment à un rapprochement avec le MOF. Seront aussi redéfinis le noyau d'UML (infrastructure) et les extensions (superstructure). Un des aspects intéressants d'UML est le concept de profil. Un profil permet d'adapter UML à un domaine qui était mal couvert. Si nous considérons qu'UML est un langage, les profils sont alors des dialectes de ce langage et ces dialectes sont fortement utilisés dans le MDA. Par exemple le profil EJB permet l'élaboration de PSM pour la plate-forme EJB. A l'heure actuelle l'OMG définit entre autres un profil pour l'EAI, un profil pour les composants d'entreprise (EDOC), un profil pour les tests, un profil pour le temps réel, un profil pour Corba et un profil pour EJB. Enfin, la dernière partie d'UML 2.0 correspond à la séparation du contenu logique d'avec la présentation physique, problème assez important qui est résolu en fournissant un méta-modèle séparé de présentation (Diagram Interchange) qu'il sera possible de modifier ou d'exploiter en utilisant des standards du W3C (XSLT et SVG notamment).

CWM (Common Warehouse Metamodel) [7]

CWM est le standard de l'OMG qui traite des entrepôts de données. Il couvre toutes les étapes nécessaires à l'élaboration, à la création et à la transformation des entrepôts de données. L'approche préconisée par ce standard pour la migration est une approche MDA. C'est-à-dire la création de modèles et leurs transformations. CWM définit les méta-modèles des principaux types d'entrepôts de données (Relationnel, Objet, XML,etc.) et propose des règles de transformation entre ceux-ci.

XMI (XML Metadata Interchange) [9]

XMI est le standard de l'OMG qui fait la jonction entre le monde des modèles et le monde XML. XMI se base sur le MOF. Il permet la génération de DTD et de schémas XML à partir de méta-modèles. L'application la plus connue de XMI est celle qui a permis la construction de la DTD UML. Cette DTD UML permet la représentation des modèles UML sous forme de documents XML et assure ainsi les échanges de modèles UML entre les différents outils du marché. Le standard XMI de sérialisation des modèles compatibles MOF est en cours de stabilisation et son utilisation devient incontournable dans les outils industriels.

D'autres propositions

L'activité essentielle de l'OMG est une activité de normalisation dans le domaine des plates-formes et des différents secteurs métier (santé, transport, énergie, commerce électronique, télécommunications, bio-informatique, etc.). Dans la période précédente, ces activités de normalisation avaient pour centre le bus abstrait Corba d'une part et IDL pour unique langage d'expression d'autre part. Aujourd'hui ces activités utilisent toutes UML, MOF et XMI pour élaborer et publier leurs accords consensuels et chaque groupe de l'OMG travaille donc à la définition de méta-modèles ou de profils spécifiques. Il est donc normal de voir se développer un nombre important de ces méta-modèles basés sur le MOF. Parmi ceux-ci, il faut faire une place importante au **SPEM** (Software Process Engineering Metamodel), précédemment nommé UPM (Unified Process Metamodel) qui définit les façons d'utiliser UML dans des projets logiciels. Le SPEM remplace donc le UP (Unified Process) en généralisant les pratiques observées autour de RUP2001 de Rational

[5] Model-Driven Architecture and Integration - Opportunities and Challenges
D. D'Souza, mars 2001
ftp://ftp.omg.org/pub/docs/ab/01-03-02.pdf

[6] Meta Object Facility (MOF) Specification
OMG/MOF, OMG Document formal/2000-04-03, mars 2000
http://www.omg.org/technology/documents/formal/mof.htm

[7] The Common Warehouse MetaModel (CWM)
OMG/CWM, OMG Document ad/2001-02-01, January 2001
http://www.omg.org/cgi-bin/doc?ad/2001-02-01

[8] Unified Modeling Language (UML), version 1.4
OMG/UML, OMG Document formal/2001-09-67, septembre 2001
http://www.omg.org/cgi-bin/doc?formal/01-09-67

ou de la "Global Service Method" d'IBM ou de la démarche Macroscope de la société DMR. Il consacre ainsi la stratégie de rupture avec OMT qui avait consisté en 1997 à ne pas se lancer dans la définition globale d'une méthode unifiée, mais beaucoup plus modestement d'un langage unifié pour la description des artefacts logiciels à objets. Cette stratégie de séparation des aspects apparaît maintenant comme excellente car elle permet de séparer le quoi (les artefacts logiciels définis en UML) du qui, quand, comment et pourquoi correspondant à l'expression séparée de la démarche de développement (Guidance, Work Product, Document, Model, ProcessRole, Activity, Step, Phase, Iteration, Lifecycle, ProcessComponent, Discipline, Process). Le tissage des aspects représentés par les deux méta-modèles complémentaires (UML et SPEM) permet donc de répondre à la question centrale de la démarche : qui fait quoi, quand, comment et pourquoi? Il est également intéressant de noter que SPEM a la particularité d'avoir été défini, pour des raisons pratiques, à la fois comme un méta-modèle MOF de plein droit et comme un profil UML, susceptible d'être directement exploité par les AGL UML du marché.

Il n'est pas possible de décrire ici tous les standards acceptés ou en cours d'élaboration en tant que profils ou méta-modèles MOF. Citons cependant les travaux en cours sur la sémantique des actions pour UML (Action Semantics) visant à normaliser, indépendamment d'un langage de programmation particulier, les différentes formes d'*exécutabilité* associables à des modèles UML.

Des défis à relever

Le MDA est un énorme chantier dont on peut raisonnablement penser qu'il va durer au moins une dizaine d'années, sinon plus. Il reste encore beaucoup de travail à faire pour clarifier la proposition [5]. Nous pouvons identifier trois axes selon lesquels il est possible de recenser les défis à relever pour le MDA :

• D'un point de vue technique, il semble que le plus gros défi à relever soit celui de la transformation de modèles, ou plus généralement de la manipulation de modèles (séparation et tissage d'aspects, fusion de modèles par exemple). Il est en effet important de pouvoir capitaliser sur ces opérations. Il faudra donc être capable de les modéliser comme nous modélisons aujourd'hui les systèmes d'information. Plusieurs initiatives commencent à émerger et proposent des approches générales pour modéliser les transformations de modèles. L'OMG a d'ailleurs publié un RFP (Request For Proposal) pour qu'un standard soit élaboré à partir de ces initiatives. La communauté W3C/XML possède avec XSLT un tel outil pour transformer les documents et les schémas. La communauté OMG/MOF/UML est en train de se définir le sien. La mise en correspondance de ces deux standards devrait être possible.

• D'un point de vue méthodologique, il semble que le plus gros défi soit de définir quelles sont les méthodes applicables au MDA. Il semble que l'évolution du cycle en "Y" en soit une. La branche de gauche correspondrait aux PIM. La branche de droite correspondrait à la description des plates-formes techniques. La jointure de ces branches correspondrait à la transformation des PIM en PSM. La branche du bas correspondrait aux PSM et à leur transformation vers du code. Approfondir le MDA revient donc à "revisiter le cycle en Y", en explicitant à chaque fois de façon précise les informations sur lesquelles on travaille par un méta-modèle spécifique.

• D'un point de vue stratégique, il paraît clair que le défi à relever consiste à savoir comment intégrer le MDA dans l'entreprise. Il faut pour cela pouvoir répondre à plusieurs questions comme savoir entre autres quels sont les avantages du MDA par rapport aux approches existantes (objet, composant, service Web), quelles sont les compétences à posséder pour utiliser le MDA, quels sont les métiers qui sont touchés par le MDA, etc. Ces questions ont pour but de permettre le calcul du retour sur investissement d'un passage au MDA. Il est prématuré de prévoir ce que sera ce ROI. Par contre on peut déjà comprendre que le coût brut de la migration de la technologie des objets et des composants vers la technologie plus générale des modèles sera au moins aussi élevé que celui du passage, dans les années 80, de la technologie procédurale vers la technologie des objets. Il faudra donc bien en évaluer les bénéfices potentiels.

Une approche élaborée par les industriels

Rien ne permet à l'heure actuelle d'affirmer que le MDA va vraiment permettre de résoudre tous les problèmes actuels du génie logiciel. Cependant, de nombreux signes sont déjà encourageants. La séparation des modèles neutres de l'entreprise d'avec les modèles de plates-formes et les modèles de couplage est une condition nécessaire à la capitalisation des acquis logiciels de l'entreprise, dans un contexte d'évolution technologique galopante. Soulignons aussi un point important : le MDA est une approche élaborée par l'OMG qui est, rappelons-le, le plus important consortium d'industriels. Il ne s'agit pas du tout ici, comme dans le cas des systèmes experts dans les années 80, d'une proposition des laboratoires de recherche au secteur industriel qui n'a que très partiellement réussi à convaincre. Le MDA est au contraire une demande élaborée par des structures industrielles et à laquelle il va falloir apporter des réponses de plus en plus précises, performantes et opérationnelles dans les années qui viennent.

Il existe déjà quelques application développées suivant une stratégie proche du MDA [10]. De plus en plus d'éditeurs affirment que leurs outils sont "MDA compliant". Le succès du MDA n'est pas garanti, mais devant le constat des problèmes complexes que l'industrie du logiciel va devoir résoudre dans les vingt prochaines années, on ne perçoit actuellement pas d'autres possibilités que de s'engager dans cette voie.

Le présent article a situé l'initiative MDA de l'OMG dans ses caractéristiques générales. Dans la suite nous proposerons une série de notes explicitant certains aspects de la technologie, de la méthodologie ou de la stratégie du MDA ainsi que des descriptions de solutions outillées, des études de cas et des témoignages de déploiement.

[9] XML Metadata Interchange (XMI) Specification v1.1
OMG/XMI, OMG Document formal/2000-11-02, novembre 2000
http://www.omg.org/cgi-bin/doc?formal/2000-11-02

[10] Wells Fargo's Business Object Services: ACase Study in Model Driven Architecture
P. Harmon, Cutter Consortium, october 2001
http://e-serv.ebizq.NET/obj/harmon_1.html

Dans ce troisième et dernier volet consacré à la programmation orientée aspect (POA), nous démontrons comment il est possible d'utiliser la POA et AspectJ pour résoudre des problèmes dans le monde réel, en modularisant les problématiques qui se recoupent.

Par Ramnivas Laddad*

La programmation orientée aspect (POA), un nouveau paradigme, permet une implémentation modulaire des problématiques qui se recoupent. Dans le premier volet de cette série traitant de la POA, j'avais défini une problématique comme suit :

"Une problématique est un but particulier, un concept, ou un centre d'intérêt. En termes techniques, un système logiciel typique comporte plusieurs problématiques : fondamentale et niveau système. Par exemple, la problématique fondamentale d'un système de traitement de cartes bancaires s'occupera des paiements, alors que les problématiques au niveau système s'occuperont du login, de l'intégrité des transactions, de l'authentification, de la sécurité et des performances par exemple. La plupart de ces problématiques, qui se recoupent pour certaines, tendent à affecter plusieurs modules. En utilisant les méthodologies de programmation traditionnelles, les problèmes de recoupement concernent de multiples modules, ce qui en résulte un système complexe à modéliser, à comprendre, a implémenter et a faire évoluer."

La POA travaille en complément, et non en remplacement, de la programmation orientée objet. AspectJ, une implémentation POA gratuite pour Java, permet de créer des systèmes logiciels simples à implémenter, à comprendre, et à maintenir. Dans le premier volet de cette série, nous avons introduit les différents concepts de la POA (*Développeur Référence v2.07*). Le second volet était lui un tutorial sur AspectJ (*Développeur Référence v2.10*). Armé de vos nouvelles connaissances sur la POA et AspectJ, cet article vous apprendra à modulariser des problématiques qui se recoupent au travers d'exemples concrets.

Bien que l'on se focalise particulièrement sur des exemples AspectJ, l'implémentation est possible dans n'importe quelle approche Aspect. En général, la POA (et c'est la même chose pour les différentes méthodologies de programmation) n'est pas centrée sur l'outil, mais sur la méthode. Les exemples sont là pour montrer les avantages que l'on à a implémenter des systèmes logiciels en utilisant la POA, que ce soit au niveau des problématiques de modularité et de recoupements, le dispersement du code, ou encore la facilité de maintenance.

Les exemples sont volontairement triviaux, pour que l'on puisse se focaliser sur l'utilisation de la POA et d'AspectJ. Dans l'exemple consacré à la gestion des threads, l'implémentation n'est que partielle. Cependant, nous nous concentrons sur les techniques de la POA. Notez également que pour le log, on se contente d'utiliser System.out.println()au lieu d'utiliser un aspect dédié, afin de rester focalisé sur la problématique principale.

Vous trouverez différents exemples dans cet article, la gestion de ressources, l'application de contraintes, les comportements basés sur des caractéristiques, ou une gestion flexible des contrôles d'accès. Nous terminerons cet article en présentant d'autres utilisations possibles d'AspectJ. A noter que les sources des exemples présentés sont téléchargeables sur notre site (voir Aller plus loin).

Modularisation d'une gestion de ressources

Ce type d'application est couramment utilisé lorsqu'il s'agit de recycler des ressources qui ont été précédemment créées, comme des threads, ou des connexions à une base de données.

Une approche objet typique nécessite que l'on fasse dès le début certains choix conceptuels. Tout d'abord, on doit spécifier quand et comment obtenir les ressources, et comment les libérer. Une adaptation du système nécessite de nombreuses optimisations, tout concevoir dès le début parait difficile. Cependant, repousser à plus tard le support requiert la modification de nombreux fichiers sources (c'est le dilemme de l'architecte, que nous évoquions dans le premier volet de cette série. De plus, les approches conventionnelles manquent d'agilité, il n'est pas évident d'activer ou de désactiver certaines optimisations. Voyons comment la POA et AspectJ peuvent nous aider dans l'implémentation d'une gestion de ressources, de manière modulaire.

Tout d'abord, mettons en place un service TCP/IP qui convertira les chaînes de requêtes

reçues en majuscules. Le serveur crée un nouveau thread lorsqu'une demande de connexion arrive, il se termine naturellement. L'implémentation suivante est en Java traditionnel, vous n'y trouverez pas d'AspectJ.

```
// UppercaseServer.java
import java.io.*;
import java.net.*;

public class UppercaseServer {
    public static void main(String[] args) throws
        Exception {
    if (args.length != 1) {
        System.out.println("Usage: java
            UppercaseServer");
        System.exit(1);
    }

    int portNum = Integer.parseInt(args[0]);
    ServerSocket serverSocket = new
        ServerSocket(portNum);
    while(true) {
      Socket requestSocket = serverSocket.accept();
      Thread serverThread = new Thread(new
            UppercaseWorker(requestSocket));
      serverThread.start();
    }
  }
}

class UppercaseWorker implements Runnable {
  private Socket _requestSocket;

  public UppercaseWorker(Socket requestSocket)
    throws IOException {
    System.out.println("Creating new worker");
    _requestSocket = requestSocket;
  }

  public void run() {
    BufferedReader requestReader = null;
    Writer responseWriter = null;
    try {
      requestReader
        = new BufferedReader(
          new
InputStreamReader(_requestSocket.getInputStream()));
      responseWriter
        = new
OutputStreamWriter(_requestSocket.getOutputStream());
      while(true) {
        String requestString =
          requestReader.readLine();
        if (requestString == null) {
          break;
        }
        System.out.println("Got request: " +
            requestString);
     responseWriter.write(requestString.toUpperCase()
        + "\n");
        responseWriter.flush();
      }
    } catch(IOException ex) {
    } finally {
      try {
        if (responseWriter != null) {
          responseWriter.close();
        }
        if (requestReader != null) {
          requestReader.close();
        }
        _requestSocket.close();
      } catch (IOException ex2) {
      }
    }
    System.out.println("Ending the session");
  }
}
```

Ensuite, voyons comment AspectJ peut nous permettre d'implémenter une gestion de threads. Tout d'abord, écrivons une classe ThreadPool, qui agira comme une pile de threads disponibles. La méthode get() permet d'extraire un thread de la pile, alors que la méthode put() permet d'en réintroduire un. La méthode put() contient également la classe DelegatingThread, qui déporte la méthode run() vers l'objet _delegatee :

```
// ThreadPool.java
import java.util.*;

public class ThreadPool {
  List _waitingThread = new Vector();

  public void put(DelegatingThread thread) {
    System.out.println("Putting back: " + thread);
    _waitingThread.add(thread);
  }

  public DelegatingThread get() {
    if (_waitingThread.size() != 0) {
      DelegatingThread availableThread
        =
(DelegatingThread)_waitingThread.remove(0);
      System.out.println("Providing for work: " +
        availableThread);
      return availableThread;
    }
    return null;
  }

  static class DelegatingThread extends Thread {
    private Runnable _delegatee;
    public void setDelegatee(Runnable delegatee) {
      _delegatee = delegatee;
    }
    public void run() {
      _delegatee.run();
    }
  }
}
```

Vous disposez maintenant des classes nécessaires à la gestion de threads. Maintenant, il nous reste à écrire un aspect qui utilise la classe ThreadPool, pour ajouter la gestion de threads à notre serveur :

```
// ThreadPooling.java
public aspect ThreadPooling {
  ThreadPool pool = new ThreadPool();

  //============================
  // Thread creation
  //============================
  pointcut threadCreation(Runnable runnable)
    : call(Thread.new(Runnable)) && args(runnable);
  Thread around(Runnable runnable) :
    threadCreation(runnable) {
    ThreadPool.DelegatingThread availableThread =
      pool.get();
    if (availableThread == null) {
      availableThread = new
        ThreadPool.DelegatingThread();
    }
    availableThread.setDelegatee(runnable);
    return availableThread;
  }

  //============================
  // Session
  //============================
  pointcut session(ThreadPool.DelegatingThread
    thread)
    : execution(void
ThreadPool.DelegatingThread.run()) && this(thread);

  void around(ThreadPool.DelegatingThread
    thread) : session(thread) {
    while(true) {
      proceed(thread);
      pool.put(thread);
      synchronized(thread) {
        try {
          thread.wait();
        } catch(InterruptedException ex) {
        }
      }
    }
  }

  //============================
  // Thread start
  //============================
  pointcut
threadStart(ThreadPool.DelegatingThread thread)
    : call(void Thread.start()) && target(thread);

  void around(Thread thread) : threadStart(thread) {
    if (thread.isAlive()) {
      // wake it up
      synchronized(thread) {
        thread.notifyAll();
      }
    } else {
      proceed(thread);
    }
  }
}
```

Examinons en détail l'implémentation :

- Le pointcut threadCreation() capture les joinpoints, en créant un nouvel objet thread qui prend en argument un objet de type Runnable.
- On interroge le pointcut threadCreation() tout d'abord pour savoir si un thread est disponible dans la pile. Si aucun thread n'est disponible, on en crée un.
- Le pointcut session() capture la méthode d'exécution run() de tout les objets ThreadPool.DelegatingThreads.
- En plaçant session() à l'intérieur d'une boucle while(true) nous assure que le service fonctionnera continuellement. Cela nous assure du fait qu'un thread, une fois créé, ne meurt jamais. Une fois que la requête à été traitée, le thread est remis dans la pile de thread, en état d'attente.
- Le pontcut threadStart() capture les appels à la méthode Thread.start(). Elle utilise la méthode isAlive() pour s'assurer que le thread a bien démarré.

Remarquez que nous n'avons fait aucune modification au fichier UppercaseServer.java. Le code pour un simple test client est dans UppercaseClient.java.

On peut utiliser la même technique pour gérer les objets de connexions à une base de données. Il suffit de capturer les joinpoints qui créent de nouvelles connexions, et leur faire utiliser une connexion existante, si une est disponible. Il faudra également capturer les joinpoints de fermeture de connexion, afin que les objets puissent être remis dans la pile.

Modularisation de contraintes de qualité

Pour implémenter des contraintes de qualité dans le code, en utilisant une approche orientée objet, on se retrouve limité à combiner des méthodes telles que la documentation, l'audit de code, etc… Des techniques exclusivement manuelles, qui limitent grandement leur efficacité, il est en effet facile de passer à coté de certaines erreurs.

Dans cette section, nous allons implémenter une gestion de contraintes de qualité pour s'assurer qu'on n'ajoute pas de doublons listener sur un même objet du modèle, et que les listeners ne traînent pas lorsque la vue qu'ils représentent est détruite.

Le modèle MVC (Modèle, Vue, Contrôleur) de Swing inclut un certain nombre de suppositions qui ne sont pas vérifiées. Tout d'abord, le modèle permet d'ajouter un listener plus d'une fois, ce qui conduit à un doublement du travail si une méthode de notification d'événement transporte une opération coûteuse. Deuxièmement, il est facile d'oublier de supprimer les listeners avant de détruire la vue. Considérez une vue dans un modèle. Vous ajoutez un listener au modèle. Le listener garde une référence vers la vue (dans un champ direct, étant donné que le listener est une classe interne de la classe vue). Maintenant, si un programmeur oublie d'enlever le listener du modèle avant de le détruire, le modèle est toujours accroché au listener qui contient l'objet vue. Etant donné que le ramasse-miettes est capable de tracer l'objet vue, il ne le collectera pas. Une telle vue va alors persister, consommant de la mémoire.

On peut mettre en place un jeu de règles afin d'éviter ces deux situations. Dans cette section, nous allons voir une manière de résoudre ce genre de problématique en utilisant l'approche aspect. Une simple compilation de votre code source, accompagnée de ces aspects suffira à effectuer les contrôles de ces règles, aucune autre modification n'est nécessaire. Cool, non ?

Nous verrons également comment utiliser UML (Unified Modeling Language) pour décrire la vue structurelle de l'aspect. Mais jetons tout d'abord un œil à la structure, la figure 1 représente la hiérarchie aspect pour la gestion d'événements.

Aspect de base : EventListenerManagement

Etant donné que l'implémentation des deux problématiques requiert la capture de joinpoints lors d'ajout de listeners au modèle, on peut partager le code au sein d'un aspect abstrait EventListenerManagement. Cet aspect contient un pointcut addListenerCall() qui capture les appels méthodes ajoutant un listener. Il utilise le pointcut modelAndListenerTypeMatch() pour permettre aux aspects qui le dérivent de restreindre les méthodes capturées par addListenerCall().

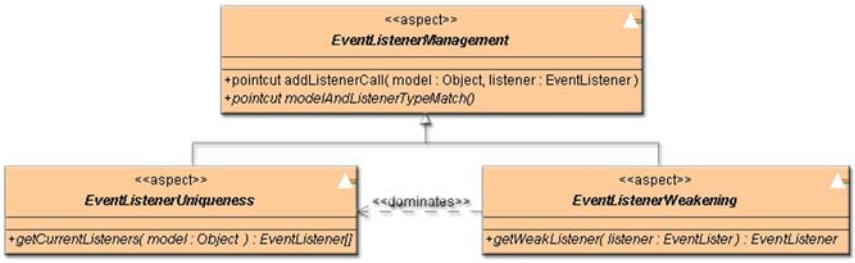


Figure 1. La structure de la gestion de listener.

```
// EventListenerManagement.java
import java.util.*;
public abstract aspect EventListenerManagement {
  pointcut addListenerCall(Object model,
    EventListener listener)
    : call(void *.add*Listener(EventListener+))
    && target(model) && args(listener) &&
       modelAndListenerTypeMatch();
  abstract pointcut modelAndListenerTypeMatch();
}
```

Implémenter la problématique d'unicité

Le fonctionnement en est simple, la problématique d'unicité, avant l'ajout d'un listener, vérifie que ce listener n'a pas été ajouté auparavant. Si ce listener est déjà présent, l'opération n'a pas lieu. Sinon, le listener est ajouté normalement.

L'aspect EventListenerUniqueness implémente ces fonctionnalités, s'assurant qu'aucun listener doublon n'est ajouté au modèle. Cet aspect abstrait déclare EventListenerManagement comme parent. Il avise le pointcut addListenerCall() afin de vérifier l'unicité du listener, en vérifiant qu'il n'est pas déjà présent dans la liste retournée par getCurrentListeners().

```
// EventListenerUniqueness.java
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.table.*;

public abstract aspect EventListenerUniqueness
  extends EventListenerManagement {
  void around(Object model, EventListener listener)
    : addListenerCall(model, listener) {
    EventListener[] listeners =
       getCurrentListeners(model);
    if (!Utils.isInArray(listeners, listener)) {
      System.out.println("Accepting " + listener);
      proceed(model, listener);
    } else {
      System.out.println("Already listening " +
        listener);
    }
  }
  public abstract EventListener[]
    getCurrentListeners(Object model);
}
```

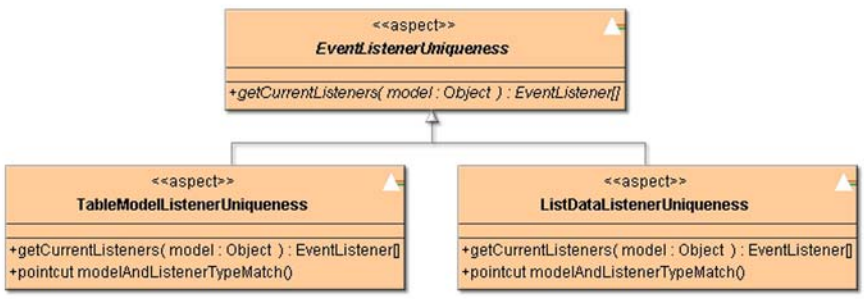
Comme indiqué sur la figure 2, l'aspect concret TableModelListenerUniqueness étend EventListenerUniqueness pour appliquer l'aspect à TableModel, et ses différentes classes. Il offre une implémentation du pointcut modelAndListenerTypeMatch() afin de restreindre le type de modèle à AbstractTableModel, et le type de listener à TableModelListener. De la même manière, on peut implémenter un autre aspect concret, ListDataListenerUniqueness, qui effectuerait le même travail pour les classes travaillant sur des listes.

```
// TableModelListenerUniqueness.java
import java.util.EventListener;
import javax.swing.event.TableModelListener;
import javax.swing.table.*;

aspect TableModelListenerUniqueness extends
EventListenerUniqueness {
  pointcut modelAndListenerTypeMatch()
    : target(AbstractTableModel) &&
       args(TableModelListener);
  public EventListener[ ]
    getCurrentListeners(Object model) {
    return ((AbstractTableModel)model)
      .getListeners(TableModelListener.class);
  }
}
```

Chaque aspect concret (dans notre cas TableModelListenerUniqueness), gère un listener associé à un type de modèle particulier. Vous pouvez créer un aspect de ce genre pour les autres modèles Swing, ainsi que pour

Figure 2. Structure d'implémentation de la problématique d'unicité pour les tables et les listes



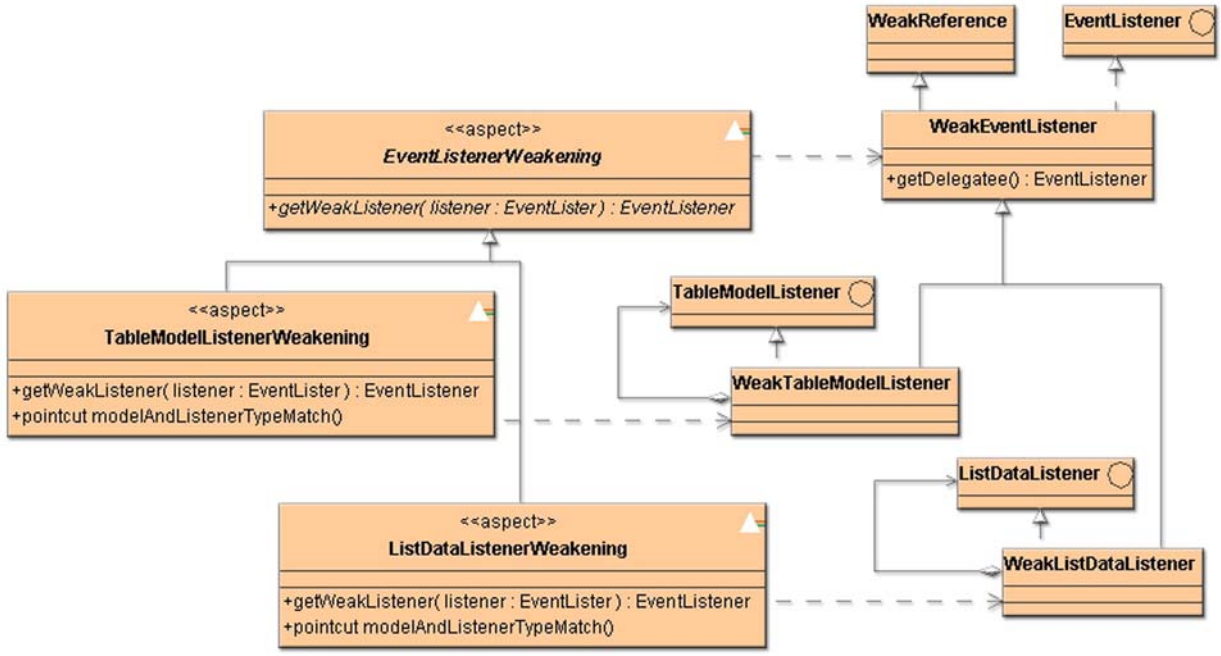


Figure 3. La structure de l'implémentation de notre problématique de gestion de vues persistantes.

vos propres modèles. Compiler ces aspects avec votre code source vous assurera qu'aucun listener n'est ajouté deux fois.

Implémenter la problématique de vues persistantes

Nous allons maintenant implémenter une problématique afin d'éviter que des vues persistent dans notre modèle, comme montré sur la figure 3. Au lieu d'ajouter des listeners directement au modèle, il est possible de l'encapsuler dans une référence en utilisant un objet WeakReference, et en l'ajoutant. Etant donné que l'objet ajouté doit être de type correct, on utilisera le motif Decorator dans une classe étendant WeakReference, et implémentant l'interface listener requise. Le Decorator implémente l'interface listener en déléguant chaque méthode à l'objet référent.

L'aspect EventListenerWeakening implémente les fonctionnalités de la problématique fondamentale, c'est-à-dire s'assurer qu'aucune vue ne persiste en mémoire après sa destruction. L'aspect abstrait déclare également EventListenerManagement en tant que parent. Il avise addListenerCall afin que ce celui-ci appelle ensuite la méthode getWeakListener().

```
// EventListenerWeakening.java
import java.lang.ref.*;
import java.util.*;
import javax.swing.event.*;
public abstract aspect EventListenerWeakening
  extends EventListenerManagement dominates
EventListenerUniqueness {
  void around(Object model, EventListener listener)
    : addListenerCall(model, listener) {
    proceed(model, getWeakListener(listener));
  }
  public abstract EventListener
getWeakListener(EventListener listener);
}
```

Le WeakEventListener décore l'EventListener en plus d'étendre WeakReference. L'aspect imbriqué RemoveGarbageCollectedListeners supprime les WeakEventListener du modèle lorsqu'il détecte qu'une référence tente d'être collectée par le ramasse-miettes. On vérifie le référent en utilisant une méthode de notification d'événement. Si vous ne souhaitez pas attendre que l'événement soit lancé après la disparition de la vue, vous pouvez utiliser une implémentation utilisant ReferenceQueues, et le thread reaper.

```
// EventListenerWeakening.java (suite...)
public abstract class WeakEventListener extends
WeakReference
  implements EventListener {
  public WeakEventListener(EventListener
delegatee) {
    super(delegatee);
  }
  public EventListener getDelegatee() {
    return (EventListener)get();
  }
  public boolean equals(Object other) {
    if (getClass() != other.getClass()) {
      return false;
    }
    return getDelegatee() ==
((WeakEventListener)other).getDelegatee();
  }
  public String toString() {
    return "WeakReference(" + get() + ")";
  }
  abstract static aspect
RemoveGarbageCollectedListeners {
    pointcut eventNotification(WeakEventListener
weakListener,
                EventObject event)
      : execution(void
WeakEventListener+.*(EventObject+))
        && this(weakListener) && args(event)
        && lexicalScopeMatch();
    abstract pointcut lexicalScopeMatch();
    public abstract void
removeListener(EventObject event,
                EventListener listener);
    void around(WeakEventListener weakListener,
      EventObject event)
      : eventNotification(weakListener, event) {
      if (weakListener.getDelegatee() != null) {
        proceed(weakListener, event);
      } else {
        System.out.println("Removing listener: " +
          weakListener);
        removeListener(event, weakListener);
      }
    }
  }
}
```

Le pointcut abstrait lexicalScopeMatch() s'assure que seules les méthodes d'un type spécifique WeakListener sont capturées. Sans ce pointcut, l'exécution des notifications de tous les WeakListener seraient capturées, par exemple celles des tableaux ou des listes. Etant donné que removeListener() nécessite des arguments de type spécifique, une exception de type ClassCastException serait lancée.

L'aspect EventListenerWeakening domine EventListenerUniqueness, qui force les appels de EventListenerWeakenig vers addListenerCall() à être appliqués avant ceux d'EventListenerUniqueness. De cette manière, on crée d'abord notre listener encapsulé, et l'unicité est vérifiée au travers de WeakEventListener.equals(). Etant donné que les listeners ajoutés au modèle sont encapsulés dans un objet WeakReference, tenter une comparaison simple au travers de la méthode equals() ne fonctionnerait pas. On pourrait éviter le problème en modifiant equals(), mais cela détruirait la nature symétrique de la fonction.

L'aspect TableModelListenerWeakening gère les listeners liés à une table. Il utilise un WeakEventListener spécialisé qui implé-

mente TableModelListener en déléguant à l'objet référent.

```
// TableModelListenerWeakening.java
import java.util.*;
import javax.swing.event.*;
import javax.swing.table.*;

public aspect TableModelListenerWeakening
extends EventListenerWeakening {
  pointcut modelAndListenerTypeMatch()
    : target(AbstractTableModel) &&
        args(TableModelListener);

  public EventListener
    getWeakListener(EventListener listener) {
    System.out.println("Weakening " + listener);
    return new
WeakTableModelListener((TableModelListener)listener);
  }
}
public class WeakTableModelListener extends
WeakEventListener
  implements TableModelListener {
  public
WeakTableModelListener(TableModelListener
delegatee) {
    super(delegatee);
  }
  public void tableChanged(TableModelEvent e) {
    TableModelListener listener =
        (TableModelListener)getDelegatee();
    listener.tableChanged(e);
  }
  static aspect
    TableRemoveGarbageCollectedListeners
    extends
WeakEventListener.RemoveGarbageCollectedListeners
{
    pointcut lexicalScopeMatch() :
        within(WeakTableModelListener);
    public void removeListener(EventObject event,
        EventListener listener) {
      ((TableModel)event.getSource())
.removeTableModelListener((TableModelListener)listener);
    }
  }
}
```

Avec tous les aspects présents ci-dessus, vous avez créé une problématique de recoupement qui évite à la fois les multiples notifications d'un même listener, ainsi que la persistance de vues pourtant détruites.

Implémentation modulaire basée sur les caractéristiques

Les opérations ayant les mêmes caractéristiques implémentent généralement des comportements communs. Par exemple, un curseur d'attente doit être affiché avant qu'une méthode particulièrement longue s'exécute. De manière similaire, tous les accès à des données critiques au niveau de la sécurité doivent être authentifiés.

Etant donné que toutes ces problématiques sont de nature à se recouper, la POA et AspectJ offrent des mécanismes pour les modulariser. Le nom d'une méthode n'indique pas forcément ses caractéristiques, il est donc nécessaire d'utiliser un mécanisme différent pour capturer ces méthodes. Pour créer un tel mécanisme, déclarez l'aspect qui gère

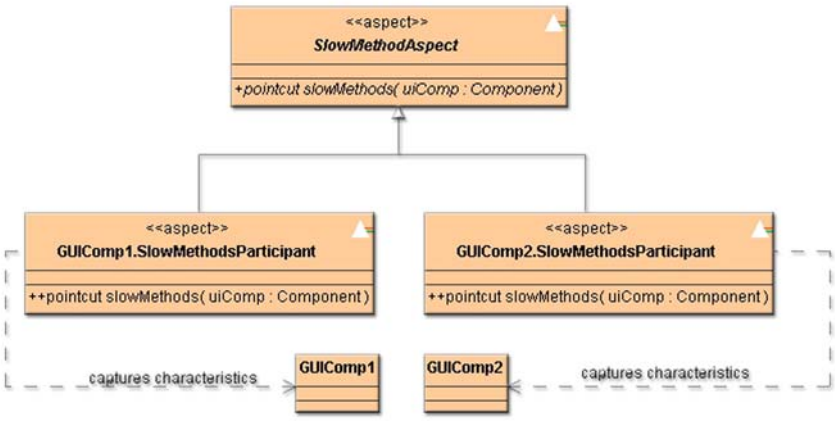


Figure 4. La structure aspect pour la gestion des recoupements basés sur les caractéristiques.

les caractéristique en tant qu'aspect abstrait. Dans cet aspect, on déclare un pointcut abstrait pour les méthodes ayant des caractéristiques que l'on souhaite considérer. Finalement, il nous suffit d'écrire l'advice qui effectuera l'opération requise.

Considérons l'implémentation de SlowMethodAspect. Il déclare le pointcut abstrait slowMethods(), et l'avise afin qu'il affiche un curseur d'attente, effectue l'opération, puis remet le curseur original, comme montré sur la figure 4. Voici le code :

```
// SlowMethodAspect.java
import java.util.*;
import java.awt.*;
import java.awt.event.*;

public abstract aspect SlowMethodAspect {
  abstract pointcut slowMethods(Component
uiComp);

  void around(Component uiComp) :
slowMethods(uiComp) {
    Cursor originalCursor = uiComp.getCursor();
    Cursor waitCursor =
Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR);
    uiComp.setCursor(waitCursor);

    try {
      proceed(uiComp);
    } finally {
      uiComp.setCursor(originalCursor);
    }
  }
}
```

Deux composants de test, GUIComp1 et GUIComp2 effectuent l'implémentation concrète de l'aspect. Par exemple, GUIComp1 contient l'aspect suivant :

```
public static aspect SlowMethodsParticipant
extends SlowMethodAspect {
pointcut slowMethods(Component uiComp)
      : execution(void
GUIComp1.performOperation1())
      && this(uiComp);
}
```

Et GUIComp2 contient l'aspect suivant :

```
public static aspect SlowMethodsParticipant
extends SlowMethodAspect {
    pointcut slowMethods(Component uiComp)
      : (execution(void
GUIComp2.performOperation1())
        || execution(void
GUIComp2.performOperation2()))
      && this(uiComp);
}
```

Comparé aux autres exemples de cet article, les classes aspect de cet exemple ne sont plus ignorant des autres aspects, elles incluent du code pour participer de manière effective à la collaboration. Cependant, cela ne va pas très loin, on se contente en effet de déclarer une méthode avec certaines caractéristiques connues. Notez que vous n'êtes pas obligés d'implémenter SlowMethodsParticipant en tant qu'aspect imbriqué. Cependant, étant donné que ces aspects dépendent complètement des classes qui les entourent, en faire des aspects imbriqués facilite la coordination du pointcut avec les futurs changements d'implémentation.

Ce type d'utilisation vous offre des interfaces de classes aspect, et vous permet de capturer les pointcuts basés sur la sémantiques ou sur les caractéristiques, ce qui est impossible à réaliser avec des pointcuts traditionnels basés sur les propriétés.

Implémenter un contrôle d'accès flexible

AspectJ permet la déclaration d'erreurs et de warning au moment de la compilation, un mécanisme qui permet de répondre aux problématiques de recoupements de manière statique. Par exemple, on peut envisager un mécanisme qui mette en place une gestion des contrôles d'accès. Les contrôles d'accès en Java (public, private, package et protected) n'offrent pas un contrôle suffisamment grand dans bien des cas. Considérez par exemple une implémentation typique de génération de produits. Souvent, vous voulez uniquement que le générateur puisse créer des objets. En d'autres termes, les classes autres que celles du générateur ne doivent pas avoir accès aux constructeurs d'aucune des classes

produites. Marquer les constructeurs avec l'accès package est la meilleure solution, cependant cela ne fonctionnera que si la factory réside dans le même package que les classes produites, une solution peu flexible.

Le mécanisme de contrôle friends de C++ permet de contrôler les accès à une classe à partir d'autres classes ou méthodes spécifiées. De même, il est possible d'implémenter une telle fonctionnalité en Java grâce à AspectJ, ainsi qu'un contrôle d'accès beaucoup plus flexible, avec une granularité plus fine.

Le code suivant déclare une simple classe Product qui dispose d'un constructeur, et d'une méthode configure(). Il déclare aussi un aspect imbriqué FlagAccessViolation, qui à son tour déclare un pointcut pour détecter les appels au constructeur en provenance de méthodes autres que ProductFactory, ou l'une de ses sous classes, ainsi qu'un autre pointcut pour détecter les appels à la méthode configure() effectuées par d'autres classes que ProductConfigurator ou l'une de ses sous classes. Pour finir, chacune de ces violations sont déclarées comme des erreurs de compilations. ProductFactory, ProductConfigurator, ou leurs sous classes respectives peuvent résider dans n'importe quel package, et les deux pointcuts peuvent spécifier le package si nécessaire. Vous pouvez même restreindre l'accès à la méthode createProduct(), par exempe, en utilisant un pointcut withincode() au lieu de within :

```
// Product.java
public class Product {
  public Product() {
    // constructor implementation
  }

  public void configure() {
    // configuration implementation
  }

  static aspect FlagAccessViolation {
    pointcut factoryAccessViolation()
      : call(Product.new(..)) &&
        !within(ProductFactory+);
    pointcut configuratorAccessViolation()
      : call(* Product.configure(..)) &&
        !within(ProductConfigurator+);
    declare error
      : factoryAccessViolation() ||
        configuratorAccessViolation()
      : "Access control violation";
  }
}
```

La classe ProductFactory appelle la méthode Product.configure(), générant une erreur au moment de la compilation avec le message d'erreur "Access control violation" :

```
// ProductFactory.java
public class ProductFactory {
  public Product createProduct() {
    return new Product();
  }
  public void configureProduct(Product product) {
    product.configure();
  }
}
```

Quelques remarques

Dans cette fin d'article, nous allons aborder quelques cas, simples, ou AspectJ montre toute sa puissance. Vous pouvez vous en servir comme exemples d'apprentissage.

Log et déboguage

Il est très facile d'implémenter en AspectJ une gestion de log et de déboguage dans un projet. Utiliser AspectJ pour ces tâches permet de journaliser différentes informations contextuelles sans toucher au code source principal.

Design by Contract

Le Design by Contract (DBC) requiert que des contrats soient explicitement respectés à différents points de l'exécution, par exemple avant et après chaque opération. On peut utiliser AspectJ pour implémenter le DBC en créant des aspects qui vérifieront les contrats à ces points d'exécution donnés. Si vous ne souhaitez pas appliquer les contrats dans un environnement de production, il suffit d'exclure les aspects DBC de la version de production.

Réaliser des design patterns

La POA permet quelques tours intéressants pour l'implémentation de certains design patterns. Par exemple, la POA permet de regarder MVC d'un autre œil. Sans la POA, les modèles MVC sont responsables de trois aspects : la gestion d'état, la gestion des listeners, et la notification de changement d'état. Avec la POA et AspectJ, on peut modulariser chacun de ces aspects, ce qui permet également d'utiliser des modèles uniquement écrits pour la gestion d'état, par exemple, Date et Point2D, dans MVC.

Créations et initialisations tardives

Il n'est pas rare d'optimiser des objets très consommateurs en mémoire en ne les créant qu'au dernier moment. Résultat, ces objets n'existent que lorsqu'ils servent pour la première fois. Comme pour toutes les autres optimisations, les instanciations tardives sont le résultat d'un profilage. Cependant, il est parfois très compliqué de changer son implémentation pour créer tardivement certains champs. Au mieux, la POO n'offre que la possibilité d'utiliser des méthodes get() pour chaque champ, et de modulariser la logique de création si besoin au sein de cette méthode. Avec cette approche, si un programmeur oublie d'utiliser cette méthode et crée un accès direct, c'est le plantage assuré. Et marquer ces champs comme privé n'aide pas plus pour les accès réalisés à l'intérieur de la classe. Même si vous vous constatez le besoin très tôt dans l'implémentation, et que le problème n'en est pas un, il est toujours intéressant de séparer les problématiques d'optimisation et d'éviter ainsi une surchage de la logique principale.

Le pointcut AspectJ get() permet de capturer les accès à un champ donné. On peut demander à un pointcut de créer un objet si le champ etait à NULL. En utilisant cette technique, la logique qui consiste à vérifier si l'objet existe avant de le créer est modularisée.

Cache

Les applications qui requièrent de hautes performances nécessitent souvent que l'on mette en cache certains objets particulièrement gourmands au moment de leur création. Par exemple, un serveur d'images créant dynamiquement des diagrammes nécessitera la mise en place d'un cache pour stocker l'image, qui sera prête à être envoyée de nouveau. AspectJ permet de séparer les problématiques liées au cache. Avec un tel modèle, on capture les joinpoints de création d'images, on regarde si une image est disponible dans le cache pour les données fournies, et si c'est le cas, on utilise cette image. Si aucune image n'est disponible, on crée alors l'image et on la stocke dans une hash table. L'implémentation du cache pourrait également employer une référence autre, afin de déterminer quelles images doivent être réellement stockées en cache, et pour quelle durée, afin de ne pas trop monopoliser la mémoire. Avec AspectJ, il est très facile de modifier après coup la politique de mise en cache, si le besoin s'en fait sentir.

Attention, ce qui suit est mon point de vue engagé sur le monde de la programmation ! La POA et AspectJ répondent à des problèmes concrets, et ceux qui s'y sont essayés en premier ont beaucoup contribué à l'évolution d'AspectJ afin qu'il soit de plus en plus en phase avec la vision globale de la POA. Mais comme tout nouveau paradigme de programmation, il faudra un certain temps avant qu'il n'atteigne son plein potentiel. Plus les développeurs utiliseront AspectJ, et plus l'expérience gagnée guidera l'évolution du langage. Ce sera très intéressant d'oberver l'adoption à grande échelle de la POA comme méthodologie de programmation, et encore mieux, d'y participer. ●

*__Ramnivas Laddad,__
est un Sun Certified Architect of Java Technology, architecte et développeur depuis huit ans. En tant qu'ingénieur chez Real-Time Innovations, il conduit le développement d'un framework de programmation par composants pour des systèmes temps réel complexes.

Traduction Guillaume Louel. "I want my AOP!, Part 2" par Ramnivas Laddad, publié par JavaWorld, copyright IDG.net, 2002. Traduit et republié avec permission. http://www.javaworld.com/javaworld/jw-04-2002/jw-0412-aspect3.html

ALLER PLUS LOIN

Réléchargez les sources des exemples présentés :
http://www.devreference.net/magazine/v216/poa3.zip

L'annuaire Web 2002 du développeur (8)

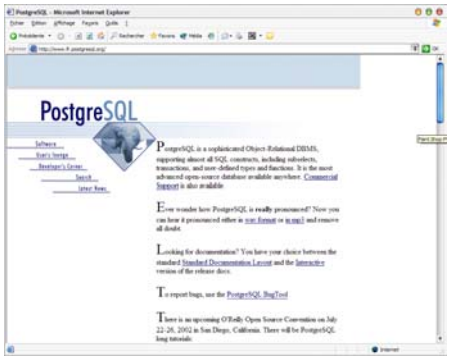
Bases de données

Frédéric Milliot
est architecte logiciel et consultant en communication. Il collabore à la presse française et américaine depuis 1989.

Dans cette huitième partie de notre sélection de sites pour développeurs, nous vous présentons les meilleurs adresses dédiées au développement de bases de données, ainsi qu'aux API et utilitaires associés.

SGBD(R)

PostgreSQL



Au même titre que MySQL, PostgreSQL s'inscrit dans un nombre croissant de projets de développement orientés Web, notamment sur des plates-formes de type FreeBSD dont il est un complément idéal. Son audience est d'ailleurs telle qu'elle suscite des conventions/conférences de développeurs, généralement sponsorisées par l'éditeur de livres O'Reilly. A quoi doit-il ce succès ? A sa robustesse, tout d'abord, qui en fait un système de choix dès lors que le projet concerné ne comporte pas de contrainte produit ou de "préconisation commerciale forte". A sa base installée ensuite, dont le dynamisme aboutit à la disponibilité permanente d'un système de qualité, bien documenté et facilement maintenable (y compris à l'extérieur). A sa compatibilité SQL enfin, qui lui permet de supporter toutes les implémentations et toutes les dérivations du langage, y compris pour les besoins les plus inattendus. Aujourd'hui en version 7.2.1, PostgreSQL est sans doute le SGBDR Open Source promis au meilleur avenir. Et si vous ne savez pas comment prononcer son nom, l'URL miroir française ci-dessous, non contente de vous l'offrir en téléchargement gratuit (code + docs), vous propose des clips WAV pour vous tirer de cet embarras. NB : compte tenu du succès de PostgreSQL, la section "API et utilitaires" de cet article liste quelques-uns des compléments les plus indispensables à une utilisation efficace.

Présentation : **
Contenu : ****
Niveau : **
http://www.fr.postgresql.org/

Firebird

Cet oiseau de feu (rien à voir avec Stravinsky ni le bolide américain) est un SGBDR puissant et complet basé sur le code Borland de l'InterBase Public License (Interbase v6.0). Implémentant SQL-92 et compatible Windows, Linux et MaxOS X (ainsi d'ailleurs, que la plupart des Unix), Firebird est donc de conception récente. Il est le fruit du travail collaboratif d'une large communauté de développeurs indépendants, et le projet

dispose d'une autorité de régulation veillant au bon développement des versions successives et au respect des règles du jeu de départ. Si vous appréciez la puissance d'InterBase 6, si vous souhaitez disposer d'un serveur et d'une API de connexion pour les front-ends clients, si comme Cognos pour PowerBuilder vous souhaitez investir vos efforts une plate-forme réellement ouverte avec accès au code, et si, ce qui est probable, vous préférez les outils réellement libres de droits y compris pour les développements commerciaux de grande envergure, alors Firebird est fait pour vous. Sur le site, assez dense de présentation, tout y est : le code, les binaires, les docs, et même les bêtas de Firebird 2.0 (avec plus de fonctions et un niveau de performance accru). PS : un driver ODBC recommandé par les concepteurs de Firebird est disponible à l'adresse :
http://www.xtgsystems.com/linux/ofbodbc/about.php.

Présentation : ***
Contenu : ****
Niveau : **
http://firebird.sourceforge.net/

Mariposa

Créé à Berkeley, Mariposa est un SGBD conçu pour les développements distribués. Son cahier des charges spécifiait dès l'origine que les WAN devaient constituer ses plates-formes d'implémentation préférentielles. Et, de fait, les concepteurs revendiquent une capacité de dimensionnement allant jusqu'à 10 000 serveurs répartis sur éventuellement autant de sites. Chaque site dispose toutefois d'une réelle autonomie locale, notamment pour le contrôle de ses propres ressources, le lancement de requêtes ou le stockage des objets. Pour maintenir cette cohérence, Mariposa ne propose aucune centralisation ni autorité pour les mécanismes de requêtes. Par ailleurs, aucun mécanisme global de synchronisation n'est ni ne doit être implémenté : cela permet, à la suite d'une modification d'objet ou de schéma fonctionnel, de maintenir constamment un temps de réponse très court. Enfin, mentionnons aussi la faculté pour un administrateur local de modifier le comportement de son site Mariposa, sachant que chaque site agit en client/serveur pour les autres sites de la base. Pour inhabituelles qu'elles soient, ces spécifications répondent plus efficacement aux besoins de gestion de données sur des plates-formes de type Web, besoins auxquels les SBGD traditionnellement conçu pour des LAN montrent rapidement leurs limites. Initié en 1996, implémentant SQL-3, le projet est aujourd'hui mature. Mariposa est libre de droits, sous les conditions habituelles de respect des mentions de copyright.

Présentation : **
Contenu : ****
Niveau : ***
http://epoch.cs.berkeley.edu:8000/mariposa/

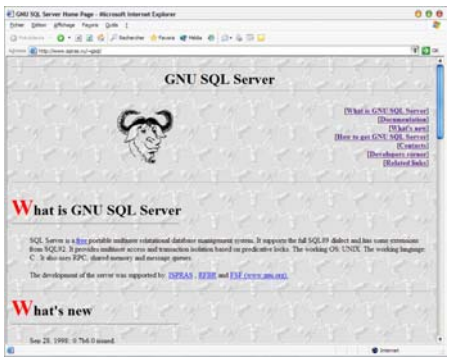
DXStore

D'origine canadienne mais anglophone uniquement, le site de DXStore vous propose un système de SGBD conçu pour les besoins les plus gourmands en terme de taille de fichier. Clairement, lorsque DXStore est utilisé en mode 64 bits interne, il y a de grandes chances pour que vous excédiez les limites logiques supportées par les OS et/ou les systèmes de fichiers cibles. Une fois configuré pour gérer des bases réparties sur un grand nombre de fichiers (en RAID 0, par exemple), DXStore est capable d'adresser 2^64 éléments de données, chaque élément pouvant atteindre la taille de 2^64 bits, et les fichiers sur laquelle la base se répartit peuvent eux-mêmes atteindre le nombre de 2^32. On est loin, très loin de la limite du Téraoctet. Le reste des fonctionnalités est à l'avenant, plus axé sur la performance et la simplicité que sur la sophistication fonctionnelle. Car DXStore servira préférentiellement à développer des système de stockage évolué ou des moteurs de recherche. Dans cette optique, toute donnée est stockée dynamiquement à partir de clés binaires, comme des BLOB, sachant que DXStore, qui bénéficie d'une faible empreinte mémoire, peut être appelé et manipulé par des scripts CGI ou intégré à des langages de scripts tels que PHP, Tcl ou Perl. Encore en version alpha bien que déjà robuste dans ses fonctions documentées, DXStore est libre de droits, sauf de celui d'oublier le copyright dans les projets commerciaux.

Présentation : **
Contenu : ****
Niveau : **
http://www.dss.bc.ca/dx/

GNU SQL Server

Plus aride que PostgreSQL, le GNU SQL Server n'implémente pas non plus la totalité de SQL-92. Reste que l'ensemble client + serveur, dont les interactions reposent sur un mécanisme RPC, bénéficie de sous-processus qui facilitent le passage simple de messages et le partage mémoire. Ce SGBD classique est beaucoup utilisé là où les besoins en matière d'administration sont importants. Les processus de compilation ou d'interprétation peuvent être lancés sur demande ou préprogrammés, tandis que certaines portions du serveur peuvent être déchargés de la mémoire disponible en cas de faible utilisation. Enfin, personne ne s'étonnera de ce que le GNU SQL reste un des systèmes favoris pour les projets dont le langage central est C, notamment pour ce qui est des types de données dont l'implémentation correspond exactement aux types standards que chacun connaît. Signalons que GNU SQL n'a pas évolué depuis 1998, ce qui ne signifie pas pour autant qu'il soit abandonné. Si vous êtes nouveau au système, vous pourrez le télécharger à l'URL ci-dessous, qui contient la plupart des documents associés. Si vous savez quoi chercher, préférez l'adresse ftp://ftp.gnu.org/pub/.



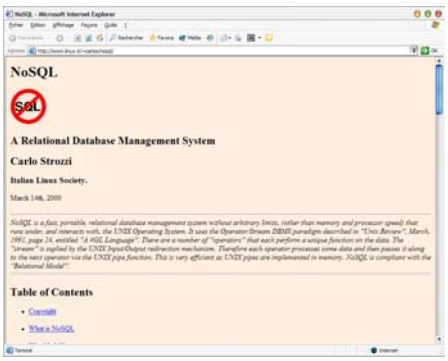
Présentation : **
Contenu : ***
Niveau : **
http://www.ispras.ru/~gsql/

NoSQL

Dérivé du SGBD RDB développé chez Rand, NoSQL est bel et bien un système de gestion de base de données, conçu à partir d'une approche au niveau noyau pour donner toute sa puissance sous Unix. Sa particularité : les données sont contenues dans des fichiers ASCII classiques. Elles peuvent ainsi être manipulées par des utilitaires et des opérateurs (ls, wc, mv…) dédiés à une fonction unique, et transmises via le mécanisme de redirection système résidant entièrement en mémoire, d'où des temps de réponse extrêmement réduits.

Fichiers "à plat" dans tous les sens du terme, les tables de NoSQL sont structurées à partir de relations entre des rangées et des colonnes. On pourrait à priori se poser la question : quel est l'avantage ? La réponse paraîtra évidente à quiconque a déjà dû manipuler de façon répétée de larges volumes de données alphanumériques pour des raisons de formatage, de correction automatique, etc. On peut se passer des front-ends qui se révèlent généralement lents et fastidieux pour ce type d'exercice.

De plus, outre le fait qu'aucune limite théorique n'existe avec ce type de fichiers (ni pour la taille des objets ou des champs, ni pour le nombre de rangées et colonnes), on peut avec NoSQL travailler sur des fichiers de données legacy (DOS, Mac, ou plus ancien encore...) en format natif, c'est à dire sans devoir gérer les logiques de traduction ANSI/ASCII toujours un tant soit peu propriétaires. D'origine italienne, accompagné d'une documentation complète, NoSQL est disponible en Open Source via une licence GPL.



Présentation : **
Contenu : ****
Niveau : **
http://www.linux.it/~carlos/nosql/

Thor

Issu du MIT, le projet Thor est un SGBD orienté-objet et distribuable à grande échelle. Autant dire une synthèse, comme dirait Michel Audiard. L'objectif, ici, est de fournir aux développeurs un substrat technique dont les mécanismes soient cachés à l'implémentation, laquelle se fonde sur une plate-forme de plus haut niveau. La première caractéristique de Thor est un langage spécifique, lui aussi développé au MIT, et baptisé theta. Theta vous offre les pré-requis indispensables pour la réalisation des concepts de Thor, à savoir des types et des hiérarchies de classes séparées, des super-types multiples, le polymorphisme paramétrique (avec pré-validation par contraintes), et le polymorphisme des sous-types. De là découle une véritable orientation objet, que l'on retrouve dans le style de codage, voire même dans l'approche analytique, et qui se traduit par une réelle efficacité aux niveaux fonctions et mémoire. Il n'est pas inutile de préciser que certains acquis de Theta sont par ailleurs applicables à C++, notamment pour ce qui concerne les changements de signatures de méthodes, à la lumière de la documentation fournie. Bref, "c'est du brutal", pour rester dans la même veine, mais, comme on s'en serait douté, là est l'avenir.

Présentation : **
Contenu : ****
Niveau : *****
http://www.pmg.lcs.mit.edu/Thor.html

DB4o

Sous une présentation inhabituelle mais plutôt élégante, le site de DB4o abrite un ensemble d'outils d'intérêt pour les développeurs Java et .NET. DB4o signifie en effet DataBase for objects, et sous ce beau programme se cache bien logiquement un SGBD objet. Performant, doté de méta-commandes destinées à simplifier autant que possible les appels fonctionnels et la manipulation des objets, DB4o ne cible aucune plate-forme en particulier, même si les auteurs vantent ses performances par rapport aux produits concurrents destinés aux PDA. Cette légère inclination provient du fait que la mise en œuvre du moteur est simple (une déclaration en CLASSPATH pour rendre l'API disponible au système et le tour est joué), et que le fonctionnement de l'ensemble est fondé sur un mécanisme d'activation qui protège la mémoire contre des allocations inutilement gourmandes. Attention toutefois, DB4o n'est ni gratuit, ni libre de droits. Il est livré sans code source, et le prix à payer dépend de la nature de l'acquéreur (petite ou grosse structure). Cela étant, pour vous donner une idée, la licence d'exploitation pour un développeur individuel est fixée à $500 par an. Notez également que DB4o met en œuvre la plupart des principes de SODA (Simple Object Database Access, dont l'API et la documentation sont disponibles gratuitement à l'adresse : http://www.odbms.org/soda.

Présentation : ***
Contenu : ***
Niveau : **
http://www.db4o.com

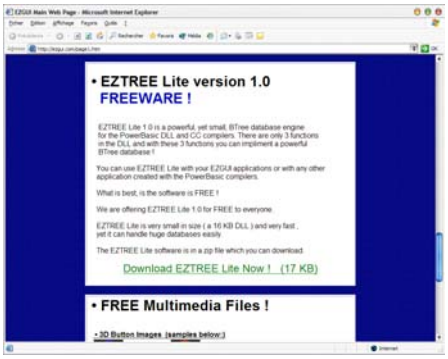
MyLittleBase

Intéressante pour bien des projets métier, myLittleBase est un SGBD embarqué conçu pour les projets Delphi, Kylix, C++ et PHP. L'originalité, ici, c'est que MyLittleBase est livré en tant que composant VCL auto-contenu. Elle s'intègre donc à tous les environnements Borland, et fonctionne sans requérir l'installation du BDE. Ses concepteurs en profitent pour vanter — à bon droit — l'intérêt d'utiliser des SGBD standalone au lieu bases de données dépendant de modules serveurs, et ce tant pour des raisons de liberté que de gain de temps en débogage. Freeware au plein sens du terme, MyLittleBase est de plus livré avec son code source et sa documentation. Si vous l'égarez ou si vous devez supporter une application en régie, un guide technique fonction par fonction et un forum d'utilisateurs sont également disponibles sur le site. Aux quatre versions téléchargeables séparément s'ajoutent une spécification du format des fichiers MLB, une interface de compatibilité pour Delphi 4/5/6, et plusieurs versions précédentes pour la maintenance de projets.

Présentation : **
Contenu : ****
Niveau : **
http://www.mylittlebase.org/

EZTree

Un petit système de type B-Tree, voilà tout ce qu'est EZTree. Certes, on est loin de la sophistication de certains autres SGBD listés plus haut, mais nous ne pouvions le passer sous silence pour deux raisons. La première, comme le rappelle en gros caractères biens moches le site de téléchargement, c'est que EZTree est 100 % gratuit, y compris en redistribution. La seconde, surtout, c'est que le moteur est livrable sous la forme d'une DLL de 16 Ko (sic). Lorsque l'empreinte mémoire est comptée, notamment pour les cas de déploiement embarqué, c'est là une véritable bénédiction. Les appels aux trois seules fonctions de EZTree peuvent être faits depuis des codes PowerBasic (il en existe encore) ou C classique. Souvenez-vous, less is more…



Présentation : *
Contenu : ****
Niveau : *
http://ezgui.com/page1.htm

API et utilitaires

The Variable Block Database

Elle avait disparue, son site s'était envolé dans des limbes, et nous l'avons retrouvée pour vous sur Planet Source Code, un site dédié au téléchargement : c'est la VBD (Variable Block Database), une bibliothèque de classes C++ assez particulière. Toujours marquée par une réelle facilité de manipulation des objets de taille variable, cette version 11.31, compatible Windows 9x et 2000/XP, Red Hat 5.2, Solaris 2.4 et HP UX 10.20 (le tout avec une complète interopérabilité), s'est enrichie d'utilitaires variés. Elle est livrée avec un pilote PostScript et un pilote HTML, et les exemples de code accompagnant la documentation (elle aussi en HTML) sont toujours plus nombreux. Les outils de manipulation d'objets textes et binaires, ainsi qu'un gestionnaire de récupération de données après incidents sont eux aussi toujours du lot. Enfin, d'un point de vue technique, signalons que la VBD supporte les conceptions OO ou relationnelles, et ce en mode local ou client serveur.

Présentation : n.a.
Contenu : ***
Niveau : ***
http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=274&lngWId=3

Metakit

Voilà une API complète qui s'adresse à tout développeur cherchant un moteur SGBD complet, performant pour des projets ne dépassant pas quelques Mo de données, et surtout capable de fonctionner dans une très petite empreinte mémoire. Ainsi, implémenté en DLL Windows, un SGBD MetaKit n'occupera qu'environ 100 Ko, la plupart des fonctions non-génériques étant appelées par l'intermédiaire de liens statiques. Techniquement, MetaKit offre ce qu'on est en droit d'attendre un moteur moderne : fichiers mappés en mémoire (si l'OS le supporte), structures de données encapsulées pour la création de documents "data-centric", sérialisation et commit/rollback, cryptage et compression, des tailles de données variables sur 1 à 32 bits, et un stockage persistant. Le tout est livré sous la forme de six classes fonctionnant comme des containers, en version C++ (API) et en binding Python et Tcl. Cette architecture a été mise en œuvre sur des plates-formes Windows, Mac, Unix, VMS, de 16 à 64 bits. MetaKit est proposé en Open Source, mais on peut disposer d'un support technique professionnel payant.

Présentation : **
Contenu : ***
Niveau : **
http://www.equi4.com/metakit/

MkSQL II

En complément de MetaKit, un utilisateur averti et qui en vaut manifestement plus de deux a développé un moteur SQL complet, venant en front-end du SGBD. Compatible SQL 2, MkSQL est livré sous la forme d'un package Python et implémente une interface DBI version 2. En tant que tel, il supporte quatre types SQL : STRING, NUMBER, BINARY et DATETIME, sachant que ROWID n'est pas supporté et que d'autres types classiques peuvent être mappés sur les précédents. Gratuit, MkSQL ne dispose pas nommément de support technique dédié, même payant ; cela

étant, il est proposé par une société, McMillan Enterprises. De même, la documentation livrée se réduit au strict minimum : d'une part, une petite introduction à la grammaire MkSQL, et d'autre part un guide technique des compatibilités et incompatibilités avec les normes SQL et la sous-couche MetaKit — ces deux documents nécessitant une certaine compétence technique pour être bien comprises. Cela étant, on est ici dans le gratuit et, nonobstant, dans l'efficace.

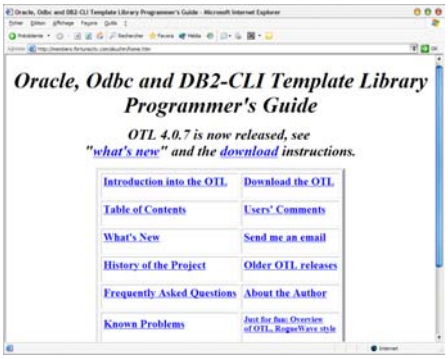
Présentation : **
Contenu : ***
Niveau : ***
http://www.mcmillan-inc.com/mksqlintro.html

DTL (Database Template Library)

On a fait dans le sobre, pour ne pas dire dans le minimal, sur le site de la DTL. Mais peu importe, le code y est et c'est bien là l'essentiel. La DTL s'adresse à quiconque souhaite utiliser des tables (record sets) ODBC comme des conteneurs STL (Standard Template Library), donc en C++. L'avantage pour le développeur, c'est une implémentation en templates de bout en bout, sans couche d'interface et sans logique de glue, toujours fastidieuse à maintenir et en tout état de cause grosse consommatrice de temps. Une fois la table connectée, tous les opérateurs STL sont disponibles, et c'est parti pour une vraie productivité. Par exemple, au lieu d'utiliser un array classique et de devoir allouer/désallouer la mémoire dynamiquement, vous pourrez utiliser un simple vector pour oublier les problème de pile et de tas, et ce faisant bénéficier des fonctions communes aux conteneurs telles que reverse, map, ou encore les différents hash_XXX. Pesant environ 700 Ko, la dernière version (3.2.4) est disponible gratuitement, à la seule condition que vous fassiez apparaître l'avertissement de copyright des auteurs sur les documentations commerciales livrées avec les produits qui en découleraient.

Présentation : *
Contenu : ***
Niveau : ***
http://www.geocities.com/corwinjoy/dtl/index.htm

OTL



Contrairement à ce que pourrait laisser entendre son nom de baptême, l'Oracle Template Library ne se limite pas aux produits Oracle (nativement en versions 7 via OCI7, 8 via OCI8, 8i via OCI8i et 9i via OCI9i). DB2 et les implémentations ODBC 2.5 et 3.0 y sont aussi, sous Windows et Unix/Linux, ce qui signifie que de nombreux SGBD sont supportés, tels PostgreSQL, MySQL, Sybase, etc. Mais supportés pour quoi faire ? Pour l'accès aux données via des templates STL combinés à des adaptateurs OTL, qui jouent ici le rôle de classes compatibles avec les frameworks C++. Le bénéfice à attendre est double. D'abord, un nombre très limité de classes permet le développement d'algorithmes complexes avec une vraie cohérence d'ensemble et une meilleure lisibilité. Ensuite, le code résultant est "traduit" en appels de fonctions natifs par les API des SGBD, d'où un niveau de performance très élevé malgré cette logique qui s'apparente à du middleware. Ajoutez à cela une portabilité complète entre les plates-formes mentionnées plus haut et une taille très compacte (moins de 400 Ko dans un seul fichier d'en-tête), et vous voilà tout à coup capable de réaliser simplement ce qui paraissait auparavant ne pas pouvoir l'être…

Présentation : *
Contenu : ****
Niveau : ****
http://members.fortunecity.com/skuchin/home.htm

Libsqlora8

On a déjà trouvé nom plus joli ! Heureusement, il n'est pas besoin de savoir le prononcer correctement pour profiter des richesses de cette API destinée aux développeurs Oracle. L'idée, ici, consiste à rendre l'accès aux données plus facile que par l'intermédiaire de l'OCI. Le premier développement a été réalisé sur une plate-forme Unix et, dixit l'auteur, la plupart des Unix sont supportés. Cela étant, les développeurs Windows ne sont pas oubliés : les sources de la bibliothèque étant intégralement gratuites, la générosité de l'auteur a poussé des utilisateurs à standardiser une méthode d'installation et même à fournir un fichier .DEF. Cette semaine, une mise à jour (v2.2.1) vient d'être rendue disponible. Pour ceux qui connaissent Libsqlora, elle ne comporte aucune nouveauté fonctionnelle mais corrige certains problèmes liés aux versions d'Oracle antérieures à 8i. Le code reste donc essentiellement le même que celui de 2.2.0, qui montrait de réels progrès dans la gestion des LOBs et gérait les mutexes Posix et Oracle. Un code, soit dit en passant, utilisé par bon nombre de professionnels facturant leurs services au prix fort… NB : si vous consultez souvent Freshmeat, sachez qu'une page dédiée à Libsqlora vous tiendra informé de la sortie de nouvelles versions en temps quasi-réel :
http://freshmeat.net/projects/libsqlora8/?topic_id=66%2C67%2C809

Présentation : *
Contenu : ****
Niveau : **
http://www.poitschke.de/libsqlora8/

OpenLink iODBC SDK

Là, on entre évidemment dans l'aride, dans la plomberie, même, pourrait-on dire, mais quand on en a besoin, ce type de ressource est une bénédiction. De quoi s'agit-il ? Comme son nom l'indique, il s'agit d'un SDK destiné au développement de pilotes ODBC. Il s'appuie sur le iODBC Driver Manager, lui aussi libre de droits, et permet le portage d'interfaces de connexions SGBD à partir d'une très large palette de plates-formes Unix. Parmi celles-ci : AIX 4.2 (PowerPC et RS/6000) et 4.3, Digital Unix v4 et v5, Solaris 2.5 à 2.8, FreeBSD 4.x (sur base Intel), HP/UX v10.20 et 11 (64 bits), etc, etc. Bref, il y a là de l'introuvable ailleurs, surtout en Open Source. A noter, le groupe est à la recherche de porteurs vers

OpenVMS, OS/400 et OS/390. Des volontaires parmi nos lecteurs.. ?

Présentation : **
Contenu : ****
Niveau : ****
http://www.iodbc.org/software.htm

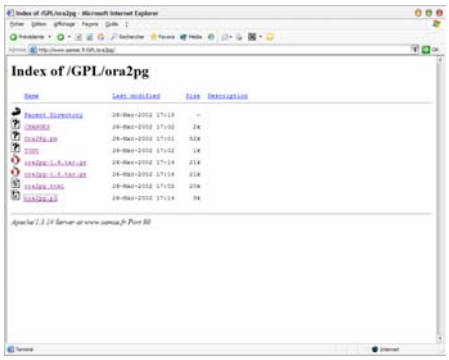
UnixODBC

Mêmes remarques que précédemment ! L'objet est ici de permettre et de standardiser l'accès en mode ODBC aux SGBD exécutés sur des plates-formes Unix. Comme l'indique la page d'accueil, le support de KDE et de GNOME est acquis. Des versions spécifiques du Manager sont disponibles en téléchargement pour s'intégrer aux bureaux desdites interfaces graphiques. En (L)GPL comme iODBC, UnixODBC bénéficie d'un support exhaustif de la part des contributeurs. Outre un historique très précis des différentes versions disponibles, il faut souligner la richesse et la qualité de la documentation, une qualité que ne reflète pas le look assez tristos du site.

Présentation : **
Contenu : ****
Niveau : ***
http://www.unixodbc.org/

Ora2pg

Pas besoin d'être devin pour comprendre qu'il s'agit là d'un utilitaire d'exportation de schémas Oracle vers des schémas PostgreSQL. L'ensemble, OO comme de bien entendu, est écrit en Perl, pour que processus soit simplifié : on se connecte d'abord à la base Oracle, le module en extrait la structure et génère ensuite un script SQL que l'on peut importer directement dans PostgreSQL. Attention, le code généré n'est pas éditable à la volée. En revanche, le dump est complet (tables, vues, séquences, indexes, permissions), et prend en compte les clés primaires, uniques et "foreign". De même, il est possible de choisir quelles colonnes seront exportées, table par table. L'auteur français, Gilles Darold, ne s'est pas ennuyé à créer un site HTML dédié. Code et documentation (en anglais) sont accessibles via ftp.



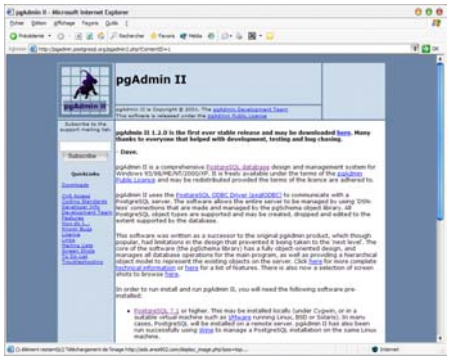
Présentation : n.a.
Contenu : ***
Niveau : ***
http://www.samse.fr/GPL/ora2pg/

OpenFTS

Basé sur PostgreSQL, OpenFTS propose une moteur de recherche Full-Text capable d'indexer des documents et d'évaluer leur pertinence par rapport à des contenus de bases de données, le tout en ligne. Son intégration à PostgreSQL permet par ailleurs l'utilisation de méta-données, de telle sorte que les recherches soit affinées, donc exploitables directement ou traitables par le SGBD de façon automatisée. Par rapport à tsearch, les possibilités sont plus évoluées. Elles incluent par exemple l'attribution d'un poids différents aux termes figurant en titre en ou gras dans les textes. Notez que les auteurs revendiquent une charge de croisière d'environ 500 000 documents, ce qui permet son utilisation sur la plupart des intranets. Livré sous la forme d'une collection de scripts Perl ou Tcl, OpenFTS est disponible en GPL. Outre le forum hébergé par le site, il peut s'accompagner d'un support technique commercial. Ca rassurera votre direction !

Présentation : **
Contenu : ****
Niveau : ***
http://openfts.sourceforge.net/

PGAdmin



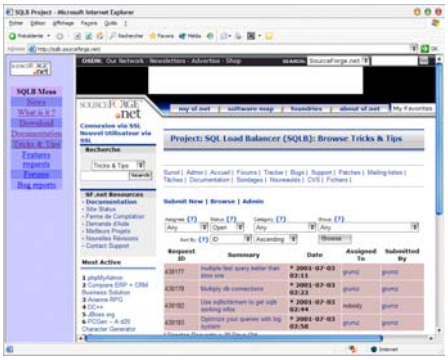
Complément idéal de PostgreSQL et comme lui libre de droits, PGAdmin est une interface unifiée de navigation dans toutes les bases d'un même serveur. Tous les objets PostgreSQL sont supportés, non seulement à la navigation mais aussi en édition, en suppression ou en création, pour autant que SQL l'autorise. Des modules d'exportation sophistiqués sont prévus pour les réponses aux requêtes les plus complexes, et cette modularité par plug-ins est étendue à certains aspects fonctionnels. Notons également une élégante gestion de la charge mémoire/réseau par un système de cache à la demande, et une interface qui fait la part belle aux barres de boutons et aux menus contextuels. D'où la question : pour quelles plates-formes ? Pour Windows, de 95 à XP.

Présentation : ***
Contenu : ****
Niveau : *
http://pgadmin.postgresql.org/pgadmin2.php?ContentID=1

SQLB

Conçu pour PostgreSQL mais aussi pour Oracle et MySQL, SQLB est un ensemble d'outils de répartition de charge (Load Balancing) fournis avec ses sources. En pratique, l'ensemble fonctionne comme un ensemble de deamons effectuant des con-

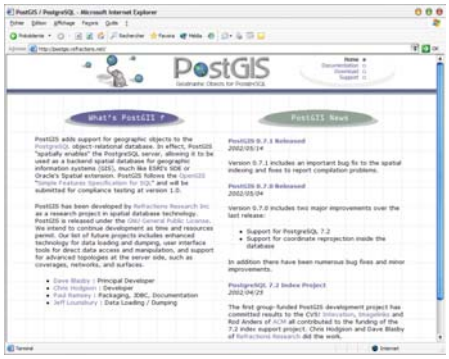
nexions multiples et permanentes. Une fois liée à la base, un programme externe, lui aussi livré en Perl et en PHP, lance des requêtes SQL aux deamons et recueille les résultats sans nécessiter de connexions/déconnexions toujours pénalisantes en terme de temps. Le gain en performance est donc plus spécialement appréciable dans deux cas de figure : lorsque l'on utilise un protocole de connexion complexe (type Oracle), et/ou lorsque la base réside sur un serveur distant. De ce fait, parce qu'il utilise les mêmes sockets Unix que PostgreSQL, l'avantage de SQLB devient nul lorsqu'il est utilisé en local. Notez également que des usages détournées de SQLB peuvent rendre de grands services. Par exemple, il est possible de l'utiliser pour effectuer des contrôles d'intégrité : SQLB vérifiera régulièrement si les requêtes en attente ne sont pas corrompues en étant stockées dans un segment de mémoire partagée. Notons d'ailleurs quatre types de logs sont possibles, incluant les erreurs, les useful information et les messages de débogage. SQLB est disponible sans frais à l'URL ci-dessous, et listera pour vous, en fonction du SGBD cible, les utilitaires complémentaires dont vous aurez besoin.



Présentation : **
Contenu : ****
Niveau : ***
http://sqlb.sourceforge.net/

PostGIS

Développé par une société commerciale et présenté sur un site très agréable, PostGIS n'en est pas moins disponible en licence GPL. Il se destine à ouvrir à PostgreSQL le vaste champ de l'exploitation géographique (GIS = Geographic Information Systems), en ajoutant au SBGD (en version 7.1.X+) des objets 3D prévus à cet effet. Equivalent de la Spatial Extension d'Oracle, comme elle compatible avec la Simple Features Specification for SQL de l'OpenGIS, PostGIS est donc conçu pour les applications de gestion de cartographie, de météo, et de bien d'autres usages de ce type.



Présentation : ****
Contenu : ****
Niveau : ***
http://postgis.refractions.net/

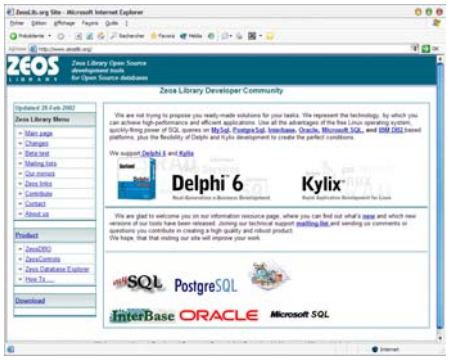
UESQLC

Capable de gérer les trois niveaux de conformité SQL, UESQLC est un compilateur SQL embarqué générant du code compatible PostgreSQL, Oracle et ODBC. Grâce à sa fonction de parsing du SQL-92, le code produit l'est sous la forme de descriptions SGML. Et justement, parce que le code produit est en SGML, il est facile de le porter vers d'autres SGBD cibles. Il n'y aura pour cela que la formalisation CDML à changer, sans qu'il faille recompiler. Toutes les sources mais aussi les fichiers binaires sont disponibles sur le site, pour un certain nombre de plates-formes (Solaris, Linux…), de même que les bibliothèques ayant servi à le coder.

Présentation : ***
Contenu : ***
Niveau : ****
http://uesqlc.dedalo-ing.com/

ZDBO

Autrefois connue sous le nom de ZDO (Zeos Database Objects), cette bibliothèque très largement utilisée dans la communauté a changé de nom pour ZDBO, mais aussi de site Web, d'où notre surprise initiale de ne pas la trouver là où nous l'attendions. ZDBO, c'est tout simplement un ensemble de composants Delphi / Kylix permettant d'accéder à de nombreux SGBD (PostgreSQL, MySQL, MS-SQL, DB2, Oracle…) sans recourir au BDE. Les API sont généralement disponibles en version bas et haut niveau, cependant que d'autres fonctions complémentaires (monitoring transactionnel, batching, édition de tables) sont également au menu. Si vous utilisez les IDE de Borland, n'hésitez pas : au fil du temps, la ZDBO devient une sorte de standard de fait. Outre la qualité des développements obtenus, sa gratuité n'y est sûrement pas étrangère…



Présentation : ****
Contenu : ***
Niveau : **
http://www.zeoslib.org/index.php

Comment le framework de la plate-forme .NET résoud-il le problème de la gestion des différentes versions des composants d'une application ?

Pierre Chalamet
est consultant et expert des architectures DNA et .NET chez Neoxia dont la mission est d'aider ses clients à maîtriser les architectures applicatives et techniques de leur système d'information. Pierre intervient à ce titre lors des phases d'architecture et de réalisation de projets ainsi que pour des séminaires techniques autour des technologies DCOM, COM+, MSMQ,

Le versionning des composants a toujours été un problème dans le monde Windows. Une fois les composants déployés, faire cohabiter des versions différentes d'une application était relativement fastidieux si certaines précautions n'étaient pas prises. Cela a été souvent décrit sous le nom de "DLL Hell". Ce phénomène apparaissait souvent lorsqu'il s'agissait de faire la mise à jour d'une application utilisant des composants partagés : DLL ou composants COM. Une application nouvellement installée fonctionnait correctement mais parfois certaines applications reposant sur les nouveaux composants partagés arrêtaient subitement de fonctionner… En théorie, ces composants associés à de bonnes pratiques ne devaient pas briser cette compatibilité, mais il se trouve que certains produits tels que Visual Basic ont donné de mauvaises habitudes en terme de versionning des interfaces de composants.

Le problème de versionning d'interfaces n'est pas spécifique à la plate-forme Windows. En effet, la problématique de versionning des interfaces partagées est d'avantage un problème de méthode. Il est donc nécessaire de comprendre que le versionning doit être utilisé dans le cadre d'une politique qualité du logiciel.

Tout le problème du versionning est qu'il est extrêmement facile de rendre une interface incompatible avec sa version précédente. Des outils tels que Visual Basic l'on très bien montré… La gestion de l'identité des interfaces ou des coclasses (implémentation sur une classe d'interfaces) est extrêmement difficile en Visual Basic sans certains artifices.

Assurer la traçabilité

Microsoft se devait donc de corriger le tir sur son terrain. Le problème du partage de composants au moyen d'interfaces connues est qu'il faut surtout préserver les contrats au niveau : des erreurs, des interfaces, des implémentations des interfaces et des dénominations des points d'utilisation (interface et implémentation).

Typiquement, avec COM cela serait : définir les HRESULT retournées par chaque opération de chaque interface ; ne jamais changer une interface publiée ; ne jamais supprimer une implémentation d'une interface sur un coclass ; ne jamais changer les identifiants d'activation pour les coclasses et interfaces.

La nouvelle plate-forme .NET de Microsoft associe simplicité et efficacité pour le développement d'application. L'objectif est de faciliter l'utilisation de la plate-forme Windows. Ce nouveau framework contient une réponse partielle aux problèmes de versionning des interfaces publiées.

Le versionning est l'acte qui consiste à marquer les binaires d'une application afin de mieux gérer les différentes versions. L'objectif du versionning est de fournir un outil pour assurer la traçabilité des différents composants d'une application. Les actions qui pourront être menées à partir de cette connaissance, c'est-à-dire quelles sont les versions intallées sur un poste en production, seront par exemple de pouvoir proposer des solutions de mise à jour d'une application à moindre coût. Une autre action pourra être de pouvoir détecter les incompatibilités éventuelles entre différentes versions d'un composant pour une exécution side-by-side. Bien entendu, utiliser le versionning de .NET sans associer un gestionnaire de sources en amoindrirait l'intérêt.

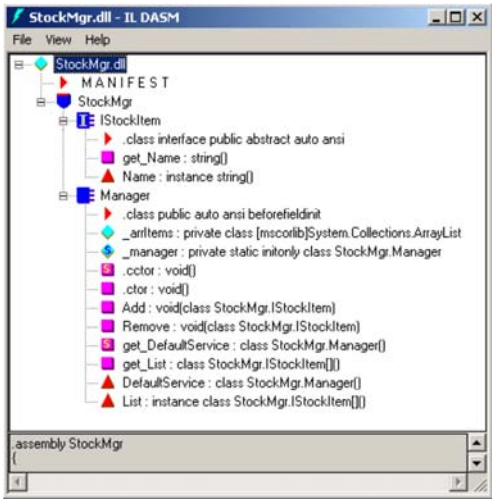


Figure 1. Vue de la structure d'une assembly (ildasm).

L'assembly, la brique élémentaire

Les applications .NET se décomposent en briques élémentaires appelées assemblies. Une assembly .NET est auto-descriptive, c'est-à-dire qu'elle contient suffisamment d'informations pour être utilisée. Une assembly est simplement une vue logique sur un ensemble de modules : cela implique qu'une assembly n'est pas simplement une DLL ou un EXE mais peut-être un ensemble de fichiers.

Une assembly est constituée de quatre éléments : le manifeste, les méta-données, l'implémentation et les ressources.

Le manifeste est présent pour la description de l'assembly et contient les informations suivantes : la dénomination (nom, version) et les dépendances avec les autres assemblies.

Les méta-données (ou métadata) contient la description de tous les types implémentés dans l'assembly :

- description des types utilisés
  - Nom, visibilité, classe de base et interfaces implémentées.
  - Membres (méthodes, champs, propriétés, évènements, types imbriqués.
- Attributs
  - Description des éléments qui permettent de modifier les types et les membres.

Les ressources quant à elles stockent des informations comme par exemple des messages d'erreurs dans une langue donnée.

Ces informations sont librement exploitables à l'aide des classes de base (BCL). Ces données sont tellement précieuses que le CLR les utilise pour alimenter ses règles de versionning et valider les types lors de l'exécution.

Une nouvelle version d'une assembly est compatible avec l'ancienne à condition que les opérations présentes sur les types exposées par l'ancienne soient présentes dans la nouvelle. Par exemple, modifier l'ordre des opérations ou ajouter des opérations sur un type ne constitue par une modification du contrat de l'interface. En revanche, supprimer une méthode en constitue une.

Identité de l'assembly

L'identité de l'assembly est probablement ce qui est le plus important car elle permettra sa localisation au moyen des règles édictées par .NET. Ses constituants sont un nom, une version, une culture et une clef cryptographique.

Un nom d'assembly permet de faire une identification partielle d'une assembly. L'usage de "partielle" provient du fait que le nom n'est pas assuré d'être unique. Par exemple, une société A peut très bien livrer une assembly nommée Isabelle alors qu'une autre société B délivre déjà une assembly sous le même nom. Le nom d'une assembly n'est donc pas suffisant pour affirmer que l'assembly à utiliser est bien celle-ci et pas une autre. Une incertitude subsiste toujours donc.

La version d'une assembly fait également partie intégrante de son identité. D'ailleurs, .NET indique une convention obligatoire concernant sa structure qu'il est important de bien respecter pour pouvoir profiter au mieux des possibilités de .NET en matière de versionning. La convention est la suivante :

<# maj>.<# min>.<# build>.<# rév>

- # maj : version majeure de l'assembly.
- # min : version mineure de l'assembly.
- # build : numéro de version d'un build.
- # rév : numéro de patch par rapport à un build.

La présence d'une version dans le manifeste de l'assembly est obligatoire.

La **version logique** d'une assembly est fourni par <# majeur>.<# mineur>. Lorsque la version logique d'une assembly change, c'est qu'il y aura probablement des soucis de compatibilités. Les changements de build ainsi que les révisions ne sont pas sensés apporter d'incompatibilités (figure 2).

La culture d'une assembly correspond à l'adaptation locale de l'implémentation. Par adaptation locale, il faut comprendre la traduction des ressources dans une langue donnée. Dans .NET, les mêmes assemblies implémentant des cultures (langues) différentes peuvent cohabiter sans aucun souci à condition de respecter certaines conditions. Par défaut, les assemblies n'ont pas de culture.

Les assemblies peuvent être signée. Par "signer", on entend que l'assembly contient une clef cryptographique. L'usage d'une telle clef ouvre des perspectives intéressantes :

- Nom de l'assembly + version + clef = identification unique d'une assembly.
- Assurer que l'identifiant de l'assembly ne peut être malencontreusement réutilisée par une autre société.
- Certifier que l'assembly à utiliser est bien celle qui a été référencée lors de la conception de l'application.
- Assurer que l'assembly ne peut être modifiée de façon malveillante.

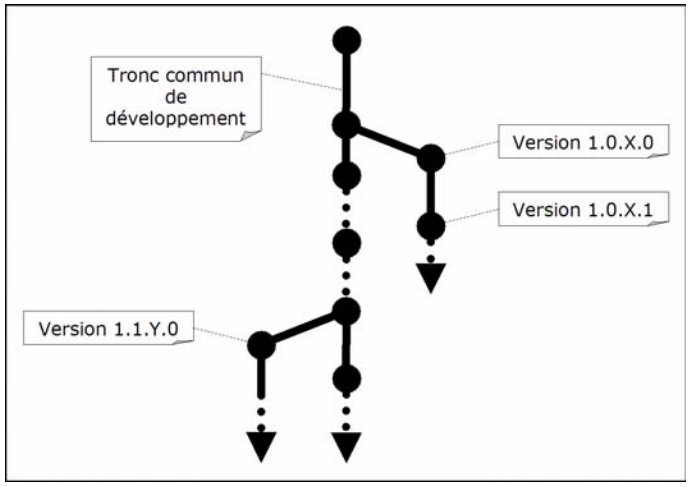
Toute assembly qui référencera une telle assembly pourra retrouver de façon certaine l'assembly.

Assembly publique et privée

Au sein de .NET, les assemblies se distinguent surtout par leurs expositions. Une assembly peut être utilisée par une seul application ou potentiellement commune à plusieurs applications.

Il y a donc la notion d'assembly privée et d'assembly publique. La différence réside dans le fait que les assemblies publiques sont signées alors que les assemblies privées ne le sont pas. Cette différence qui pourrait sembler minime est importante. En effet, le versionning n'est actif que pour les assemblies signées. C'est-à-dire que si une assembly

Figure 2. Tronc et branches des différentes versions d'une application.



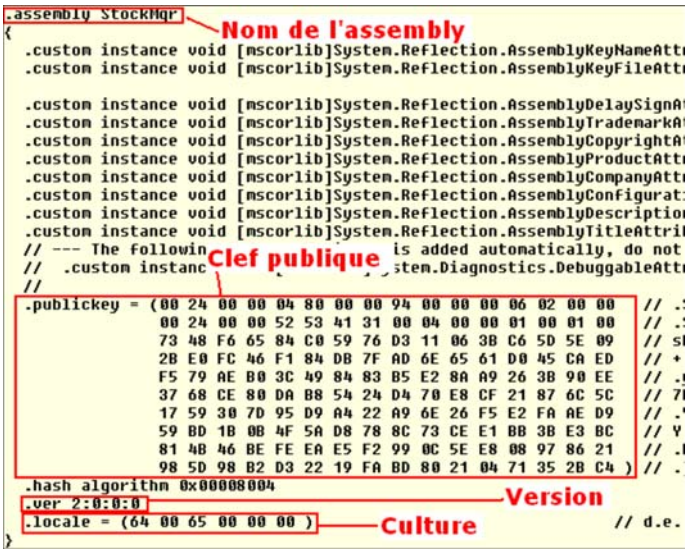


Figure 3. Extrait du manifeste d'une assembly (ildasm).

demande une version précise d'une assembly privée (donc non signée) alors la version sera complètement ignorée : .NET prendra en parcourant les chemins de dépendances de l'application la première assembly présente correspondant au nom de l'assembly… Il faut donc faire particulièrement attention à ne pas avoir deux assemblies avec le même nom dans les chemins de dépendances de l'application.

Pour une assembly publique le mécanisme est totalement différent. Une résolution d'assembly ne pourra être honoré que si le framework arrive à trouver une version égale ou équivalente de l'assembly demandée. Il y a eu un changement dans la stratégie du versionning entre la beta1 et la beta2 de .NET.

Physiquement, une assembly privée se trouve toujours dans l'un des chemins de résolution de l'application. Une assembly publique est soit dans l'un des chemins de résolution de l'application soit dans le cache global des assemblies (GAC).

Résolution des assemblies

Lorsqu'une assembly référence une autre assembly nécessaire pour l'exécution d'une application, .NET fait ce qui est appelé une résolution d'assembly. L'action consiste en fait à chercher quelle est l'assembly qui devra être chargée pour que l'application continue son exécution. Une assembly signale une référence à une autre assembly de deux manières : par son manifeste ou par un chargement explicite d'une assembly.

Peu importe la méthode de chargement utilisée, le résultat étant le même : il y a soumission d'une demande de résolution d'assembly au framework. Celle-ci se fait en donnant les critères suivants :

- Nom de l'assembly.
- Version.
- Culture.
- Clef publique.

C'est en fait l'identité complète de l'assembly à rechercher.

L'usage d'un ou de plusieurs de ces attributs (avec au moins le nom) est appelé un nom partiel. Le nom partiel contenant au moins le nom de l'assembly. Au contraire, un nom complet utilise tous les attributs.

Avec les éléments fournis, .NET appliquera des règles de versionning afin de déterminer quelle est l'assembly finale à utiliser. En effet, comme il sera vu plus loin, il est possible en fonction de l'évolution de l'application ou des assemblies de spécifier des stratégies pour la résolution. Mais attention, ces stratégies ne seront valables que sur des assemblies signées…

Nous allons voir maintenant, comment il est possible d'exploiter la mécanique de versionning de .NET, afin de la mettre à profit dans ses projets. Ce regard sur le versionning se fera grâce à un exemple d'une petite application de gestion de stock. Cette application, StockManager, utilisera dans un premier temps des assemblies privées, mais puisque les assemblies privées ne sont pas versionnables, des assemblies publiques seront utilisées par la suite.

Versionning des assemblies privées

StockManager sera une application de test sur laquelle sera expérimenté le versionning de .NET autour de deux scénarios : livraison d'une version et livraison d'un patch. Voir à la fin de l'article pour les sources de cette petite application.

StockManager version 1.0.0.0

La première version de StockManager qui a été développé est passée en production dans la version 1.0.0.0.

StockManager version 1.0.0.1

Au cours des jours suivant sa mise en production, un problème de fiabilité a été détecté. Dans le cadre d'un accès par plusieurs threads à la classe Manager, des problèmes de corruption des données peuvent survenir mettant ainsi en péril le bon fonctionnement des applications. Pour cela, l'équipe de développement fait un QFE (Quick Fix Engineering ou patch) par rapport à la version 1.0.0.0. Il faut noter que ce patch sera issu des sources de la version 1.0.0.0 et non des sources que l'équipe de développement possède actuellement qui sont certainement à un stade plus avancé que la version 1.0.0.0. Un QFE est un patch généralement temporaire le temps qu'une autre version soit disponible. Le patch est installé dans le répertoire de l'application, les assemblies n'étant pas signées.

On voit alors que l'application utilise la nouvelle version automatiquement et cela sans configurer le moindre fichier ce qui potentiellement dangereux.

Versionning des assemblies publiques

Le versionning des assemblies publiques est plus délicat puisqu'elles sont potentiellement utilisées par une ou plusieurs applications. Un tel versionning demande une grande rigueur au niveau de la compatibilité. C'est pour cela que par défaut lors de la résolution de telles assemblies, .NET recherche uniquement l'assembly demandée : il n'y a en aucun cas de promotion de version. Si une application demande une version précise d'une assembly publique, la demande ne sera honorée que si une version équivalente est trouvée. Cela évite qu'une application utilise une nouvelle version qui ne lui est peut-être pas destinée (dans le cas d'une installation side-by-side de deux versions différentes d'une même application).

Le problème va donc commencer lors des mises à jour des composants d'une application. Il va falloir indiquer par un moyen ou un autre que l'application doit utiliser les nouveaux composants. En fait, pour qu'une application utilise une nouvelle version d'une assembly publique, il faut que cela soit explicitement indiqué. A cette fin, il faut utiliser des fichiers de stratégie (policy file) afin de donner des règles de versionning à .NET.

Dans .NET, il est possible de spécifier qu'une certaine version d'assembly doit être promue. Cette étape est soit un acte volontaire de la part de l'éditeur soit de l'application. C'est pour cela qu'il existe deux scénarios principaux :

Scénario pour l'éditeur

L'éditeur souhaiterait faire en sorte que toutes les applications utilisant une assembly utilisent une nouvelle version (car par exemple elle corrige des bugs). Il faut noter que cette version est compatible avec l'ancienne.

Pour cela, lorsque l'éditeur va livrer la nouvelle version de son assembly, il fournira également un fichier de stratégie d'éditeur (publisher policy). Ce fichier précise comment la promotion de version doit s'effectuer. La redirection dans ce cas s'effectue d'après la version logique de l'assembly.

Scénario pour l'application

Un éditeur a fourni une nouvelle version de son assembly à une société mais sans fournir de fichier de stratégie d'éditeur. Mais l'administrateur de l'application souhaite profiter malgré tout de cette mise à jour sans installer une nouvelle version de l'application. Pour cela, il suffit de géné-

rer un fichier de stratégie pour l'application afin de préciser quelle doit être la nouvelle assembly à utiliser.

Pour illustrer les possibilités du versionning des assemblies publiques, l'application présentée précédemment va être de nouveau utilisée.

StockManager version 2.0.0.0

La société qui avait développé la version précédente a livré la version 2.0 à ses clients. Cette version brise toute compatibilité avec les versions précédentes. Cette nouvelle version est présente sous la forme de deux assemblies. L'une contient l'interface graphique de l'application, l'autre le service métier de gestion des stocks. A l'avenir, ces deux assemblies peuvent être mis à jour indépendamment.

Le service de gestion des stocks est également commun à toutes les applications. Le service sera installé dans le GAC de la machine hôte.

StockManager version 2.1.0.0

Cette nouvelle version apporte des "améliorations" au niveau du service de gestion des stocks. Cette version apporte de nombreuses améliorations pour les nouvelles applications basées sur ce service mais tout en conservant la compatibilité avec les anciennes applications.

Pour faire la mise à jour, l'éditeur est obligé d'éditer une stratégie afin que les anciennes applications utilisent le nouveau moteur. La stratégie de l'éditeur est de promouvoir la version 2.0.0.0 du service vers la version 2.1.0.0. Pour cela, l'éditeur va construire une assembly représentant la stratégie de son versionning à savoir faire une promotion de version pour toutes les applications (listing 1).

L'attribut oldVersion de <bindingRedirect> permet également de spécifier un intervalle de version source comme suit :

```
<bindingRedirect
oldVersion="2.0.0.0-2.0.8.0"
newVersion="2.1.0.0" />
```

Comme on le voit, il faut préciser le nom de l'assembly ainsi qu'un token. Pour rappel, un token est la version réduite d'une clef publique. L'assembly de stratégie est ensuite assemblée dans une assembly et signée au moyen de la clef cryptographique qui a servi initialement. Dans ce scénario, c'est donc bien l'éditeur qui fixe sa politique de versionning et personne d'autre : en effet, tant que la clef cryptographique utilisé par l'éditeur restera privée, personne ne pourra générer de telle stratégie.

Reste à comprendre comment se fait la promotion de version d'une assembly. Lorsqu'il y a demande d'une résolution d'assembly, .NET teste avant tout la présence de stratégie d'éditeur pour une assembly. Pour cela, il utilise le nom de l'assembly, sa version logique ainsi que sa clef publique. Pour tester sa présence, .NET essaiera de trouver dans le GAC la stratégie dénommée comme suit :

```
policy.<version majeure>.<version
mineure>.<nom de l'assembly>
```

C'est pour cela qu'il faut packager une stratégie d'éditeur sous la forme d'un module d'une assembly. L'assembly devant respecter la convention de nommage donnée précédemment. Pour générer une telle assembly, il faut utiliser l'assembly linker (al.exe) livré avec le SDK, Visual Studio ne pouvant générer de telles assemblies.

D'un autre coté, il peut s'avérer que l'éditeur ne propose pas de stratégie pour une promotion automatique de version. Il est donc possible dans .NET de spécifier qu'une application devra utiliser une autre version d'une assembly. Il faut savoir que cela est fait sous l'entière responsabilité de l'administrateur de l'application. Cela est possible en créant un fichier dans le même répertoire de l'application dont le nom est celui de l'application avec .config à la fin (par exemple Test2.exe.config pour l'application Test2.exe). Pour reprendre le même cas que précédemment, il suffit de mettre dans le fichier de configuration le <bindingRedirect> et de supprimer le fichier de stratégie d'éditeur tout simplement.

Les deux scénarios principaux ont donc été vus : d'une part le point de vue de l'éditeur qui est capable de certifier qu'une promotion de version est sans danger et d'une autre part l'administrateur de l'application qui souhaite forcer l'utilisation d'une nouvelle version. Malgré tout, on voit que l'éditeur peut forcer toutes les applications à utiliser la nouvelle assembly et provoquer malencontreusement des problèmes de compatibilité qui provoqueront un dysfonctionnement de certaines applications. Pour y remédier, l'administrateur d'une application peut indiquer que l'application ne souhaite pas utiliser la promotion de version automatique pour une assembly donnée (listing 2).

Une application qui utilise dynamiquement des assemblies au moyen d'Assembly.Load(), aura sûrement intérêt à utiliser un alias pour la dénomination de ces assemblies. Pour cela, .NET permet d'associer un nom partiel à un nom complet d'assembly (listing 3).

Bien entendu, l'ordre de réécriture des règles ne se fait pas n'importe comment. .NET applique donc les stratégies dans l'ordre suivant :

1. Au niveau de la machine.
2. Au niveau de la publication (stratégie d'éditeur).
3. Au niveau de l'application.

L'ordre de réécriture des règles se fait dans l'ordre indiqué, la stratégie au niveau de l'application étant toujours plus forte que les stratégies au niveau de la machine. Les stratégies éditeurs et application différaient dans leur packaging (assembly ou fichier .config) mais les stratégies application et machine sont identiques. Le fichier servant à la configuration des assemblies au niveau de la machine est situé ici :

```
%SYSTEMROOT%\Microsoft.NET\
Framework\%FRAMEWORKVERSION%\
Config\machine.config
```

.NET utilise un format particulier pour la désignation des assemblies dans les fichiers de stratégies ou avec la BCL. L'objectif de

Listing 1. Exemple de promotion d'une version d'assembly en utilisant <bindingRedirect>.

```
<?xml version="1.0"?>
<configuration>
 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
    <assemblyIdentity name="StockMgr"
   publicKeyToken="2a25df34ec4ba7e0" />
    <bindingRedirect oldVersion="2.0.0.0" newVersion="2.1.0.0" />
   </dependentAssembly>
  </assemblyBinding>
 </runtime>
</configuration>
```

Listing 2.

```
<?xml version="1.0"?>
<configuration>
 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
    <assemblyIdentity name="StockMgr"
     publicKeyToken="2a25df34ec4ba7e0" />
    <publisherPolicy apply="no" />
   </dependentAssembly>
  </assemblyBinding>
 </runtime>
</configuration>
```

Listing 3. Exemple en utilisant <qualifyAssembly> :

```
<?xml version="1.0"?>
<configuration>
 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <qualifyAssembly partialName="StockMgr" fullName=
"StockMgr,version=2.0.0.0,publicKeyToken=2a25df34ec4ba7e0,culture=neutral"
/>
  </assemblyBinding>
 </runtime>
</configuration>
```

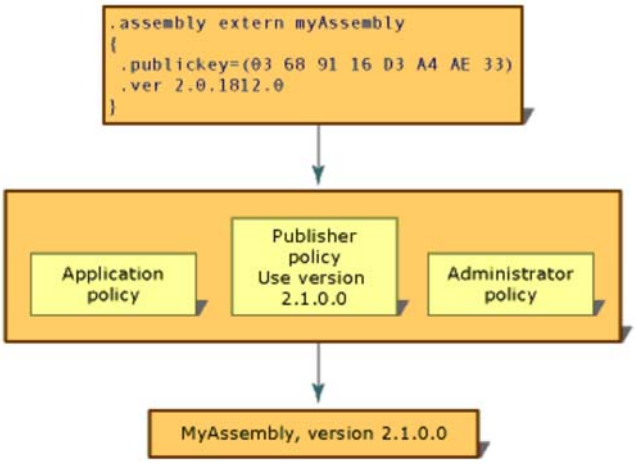


Figure 4. Ordre de réécriture des règles.

```
<?xml version="1.0"?>
<configuration>
 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
    <assemblyIdentity name="StockMgr"
     publicKeyToken="2a25df34ec4ba7e0" />
    <codeBase version="2.0.0.0"
     href="HTTP://www.neoxia/download/StockMgr.dll" />
   </dependentAssembly>
  </assemblyBinding>
 </runtime>
</configuration>
```

Listing 4. Localisation d'une assembly.

ce format est de décrire l'identité d'une assembly sous une norme.

Exemple de nom complet pour une assembly :

```
StockMgr,Version=2.0.0.0,
Culture=neutral,PublicKeyToken=
2a25df34ec4ba7e0,Custom=null
```

Exemple de nom partiel pour une assembly :

```
StockMgr,Version=1.0.0.0
```

Le déploiement

Les assemblies sous .NET sont auto-descriptives. Grâce à cette capacité, les assemblies n'ont pas besoin de modifier la base de registre pour pouvoir être utilisées. Comme vu précédemment, l'utilisation de stratégies de configuration permet un bien meilleur contrôle qu'auparavant.

Le déploiement consiste à définir comment seront localisées les assemblies par rapport à une application. Il prend également en compte la façon dont la mise à jour sera faite. Le déploiement fait partie d'une politique de mise à jour.

Définir l'accès aux assemblies

Comme vu précédemment, il existe deux sortes d'assemblies dans .NET : les assemblies privées et les assemblies publiques. La seule distinction étant le fait qu'elles soient signées ou non. La recherche des assemblies privées ou publiques n'est pas différente. Elle se fait par rapport aux répertoires configurés pour une application après la promotion de version (figure 5) :

- Rechercher dans le GAC si l'assembly demandée est signée.
- Rechercher dans les chemins spécifiés par <codeBase>.
- Rechercher dans les chemins de déploiement standard de .NET.
- Rechercher dans les chemins spécifiés par <probing>.

Après un échec d'une recherche dans le GAC, .NET recherche dans les répertoires spécifiés par <codeBase>, ceci permet de spécifier la localisation d'une version précise d'une assembly. Un bon usage de <codeBase> est de spécifier un serveur HTTP (c'est le seul protocole supporté) où pourront être téléchargé les dernières versions des assemblies (listing 4). Une des limitations de <codeBase> est qu'il faut fournir explicitement la version de l'assembly à atteindre.

Avec IIS, il faut mettre le droit d'exécution "script only" sur l'application Web qui servira de racine pour la distribution des assemblies. De plus, si l'assembly est déployé par HTTP alors il faudra se plonger dans la sécurité de .NET. Cela sera montré un peu plus loin.

Après un échec avec <codeBase> .NET recherche dans les chemins de déploiement standard d'une application, à savoir les chemins suivants et dans cet ordre :

- [Base] / [Nom de l'assembly].DLL
- [Base] / [Nom de l'assembly] / [Nom de l'assembly].DLL
- [Base] / [Nom de l'assembly].EXE
- [Base] / [Nom de l'assembly] / [Nom de l'assembly].EXE

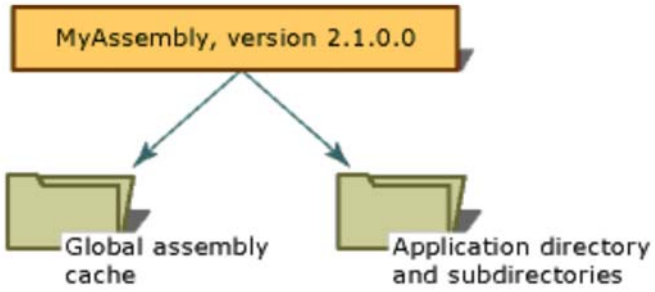
Avec [Base] ayant pour valeur [Base de l'application]. Si une culture est fournie alors la recherche s'effectue dans les mêmes répertoires sauf que [Base] vaut à [Base de l'application] / [Culture].

Et enfin, seulement après un échec d'une recherche dans les chemins de déploiement standard, .NET recherche dans les répertoires spécifiés par <probing>. L'usage de <probing> permet de spécifier des chemins qui pourraient éventuellement contenir des assemblies :

```
<?xml version="1.0"?>
<configuration>
 <runtime>
  <assemblyBinding xmlns=
"urn:schemas-microsoft-com:asm.v1">
   <probing privatePath="v2.1.0.0"/>
  </assemblyBinding>
 </runtime>
</configuration>
```

Les chemins spécifiés au moyen de <probing> sont des chemins relatifs par rapport à l'application. Plusieurs chemins pouvant être spécifié au moyen du séparateur point-virgule.

Figure 5. Recherche des assemblies.



La sécurité des déploiements

Les déploiements soulèvent souvent des problèmes de sécurité, car il faut limiter l'utilisation des applications aux seules personnes autorisées. .NET est un framework qui est fortement sécurisé et impose donc des contraintes d'exécution. Le CAS (Code Access Security) est présent afin de gérer les droits d'exécutions de toutes applications sur la plate-forme .NET.

La plupart des classes de la BCL imposent des limitations au niveau de l'exécution. Par exemple, pour ouvrir une fenêtre sur un poste client, il faut que l'application qui s'exécute possède certains droits (ici FullTrust c'est-à-dire confiance totale). Les droits accordés à une application sont résolus avant toute exécution de l'application avec l'aide des preuves apportées par l'environnement d'exécution (Internet Explorer, Loader…).

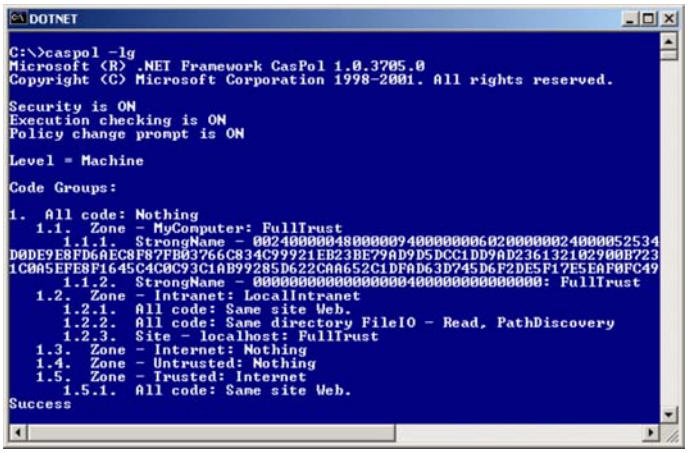


Figure 6. Groupes présents sur une machine (caspol).

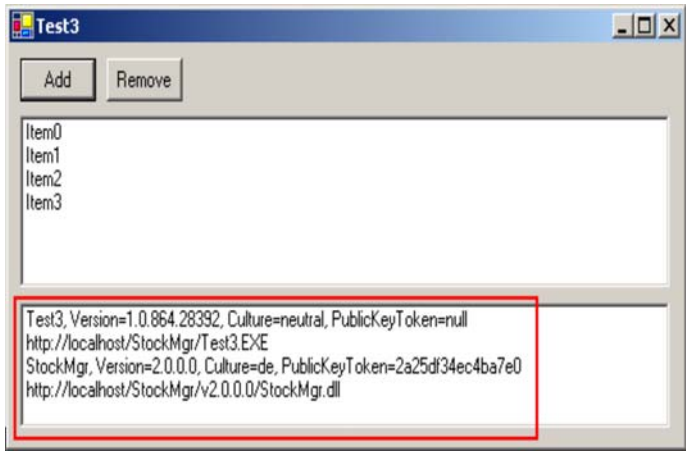


Figure 7. Exemple de déploiement d'une application par http (Test3).

La sécurité dans .NET est organisée par en groupes auxquels sont associés un ensemble de permissions. Ces groupes possèdent des caractéristiques qui permettent de les identifier. La liste des groupes est obtenu en exécutant la commande "caspol -lg" dans une console (figure 6).

A chaque groupe, un ensemble de permissions par défaut est associé. .NET en définit un certain nombre :

- **Nothing** : aucune permission. Aucun code ne peut s'exécuter.
- **Execution** : permission pour l'exécution mais impossibilité d'utiliser des ressources protégées.
- **Internet** : permissions par défaut pour un contenu d'une provenance inconnue.
- **LocalIntranet** : permissions par défaut pour un contenu en provenance de l'entreprise.
- **Everything** : toutes les permissions standard sauf celles qui permettent d'éviter les étapes de vérification.
- **Fulltrust** : toutes les permissions.
- **SkipVerification** : autorise la suppression de l'étape de vérification.

Cette liste est le jeu par défaut des groupes de .NET mais bien entendu, il peut être augmenté au besoin.

L'attribution d'une assembly à un groupe se fait en fonction de preuve que l'environnement d'exécution peut apporter : répertoire d'installation de l'application, clef publique d'un éditeur, URL... Par exemple, si une assembly est téléchargée à l'aide de <codeBase> à partir d'un site Internet, le groupe qui lui sera associé par défaut sera Internet. Hors, par défaut, le groupe Internet ne possède pas beaucoup de droits pour l'exécution. Il faudra donc lui en attribuer de façon chirurgicale afin de pouvoir exécuter des assemblies téléchargé de cette façon.

Voici un exemple qui attribut tous les droits aux assemblies téléchargé à partir du site www.neoxia.com :

```
caspol -ag 1.2 -site www.neoxia.com
FullTrust
```

Attention! Ceci est un exemple et ne doit pas être fait à la légère. Il est particulièrement recommandé de s'intéresser à d'autres scénarios comme le "partially trusted code".

Déployer par Internet

Maintenant que la sécurité a été effleurée, il est possible d'envisager un déploiement par Internet. Pour cela, il faudra tout d'abord installer une application exécutable .NET (qui n'est rien d'autres qu'une assembly) dans un répertoire d'une application Web IIS. Le point à surveiller au niveau de la configuration de cette application Web est qu'il faut attribuer le droit d'exécution "Script only" à l'application. Il faudra également veiller à autoriser la lecture du répertoire bien évidemment.

Il ne reste alors qu'à pointer vers l'URL de l'exécutable pour que .NET télécharge l'application et toutes ses dépendances au fur et à mesure de leur utilisation. Afin d'améliorer le démarrage de l'application, un raccourci vers l'application .NET pourra être déposé sur le bureau de l'utilisateur. Dorénavant, les mises à jour pourront être faites sur le site même et cela sans déployer la moindre assembly sur le poste client mais attention, cela demande une intervention : donner les droits nécessaires à l'exécution des assemblies sur le poste client (figure 7).

Exploiter le versionning de .NET

La gestion des dépendances d'une application vis-à-vis de ses composants est un problème relativement délicat surtout dans un contexte de mise à jour partiel. Néanmoins, il est souhaitable de n'utiliser autant que possible que des assemblies privées. En effet, il est préférable qu'une application n'utilise pas de composants partagés qui sont souvent sources d'ennuis lors des mises à jour.

D'un autre coté, utiliser des assemblies publiques permettra de renforcer la compatibilité entre applications qui doivent partager des composants. Cela est typiquement le cas des éditeurs de logiciels. Une politique de versionning stricte doit dès lors être employée pour les mises à jour : tests de non régression, politique de gestion de sources et suivi des modifications des interfaces publiques pour gérer les problèmes de compatibilités, gestion des dépendances, gestion des droits d'usages… Sinon ce n'est pas le "DLL Hell" qui surviendra mais bel et bien le "Assembly Hell" !

Un point important à surveiller dans ce cadre est la gestion des erreurs. Une opération sur une classe étant capable de lever des exceptions, il est regrettable que .NET n'associe pas les exceptions possibles aux signatures des interfaces comme le fait Java. En effet, cela permettrait dans certains cas d'assurer une meilleure compatibilité et un contrôle plus strict lors des compilations.

Savoir utiliser le versionning de .NET est un point crucial pour pouvoir diffuser des applications fiables et évolutives. D'ailleurs, il est utilisé à de nombreux endroits dans le framework : interopérabilité COM, composants COM+ avec les ServicedComponent, remoting, sérialisation... pour ne citer que ceux là !

ALLER PLUS LOIN

Téléchargement des sources de StockManager : http://www.neoxia.com

Schéma pour la configuration des stratégies : ms-help://MS.VSCC/MS.MSDNVS/cpgenref/html/gngrfruntimesettingsschema.htm