La suite VisualAge for Java entreprise d'IBM

VisualAge semble parfaitement adapté pour le développement sous Windows de grosses (même très grosses) applications en Java. Ses défauts ne l'empêchent pas de se classer parmi les meilleurs outils du genre.

Frédéric Mazué
frederic.mazue@wanadoo.fr

Nous testons aujourd'hui Visual Age pour Java d'IBM, Edition Entreprise, version 3.5 pour Windows. La machine de test est un PII 233Mhz avec 256 Mo de mémoire vive. Le système d'exploitation est Windows 98. Lorsque vous acquérez VisualAge pour Java vous obtenez entre autres :

- Un environnement de développement intégré pour la création *d'applets* et d'applications Java, de *servlets, d'Entreprise Java Beans* et permettant l'écriture d'interfaces IDL.
- Un débogueur local et distribué. Le débogueur distribué tourne sur Windows NT, OS/2, AIX, AS/400, Solaris et OS/390.
- Un support de développement coopératif.
- DB2 pour Windows et Linux.
- Des outils d'accès aux bases de données *via* SQLJ.
- Des outils d'accès aux serveurs d'applications WebSphere et Domino.
- Des outils d'optimisation du code Java pour AS/400 et OS/390.
- Des outils originaux tels que Persistence Builder et C+ + Access Builder.
- Net.data.

En revanche vous n'obtenez pas :

- Un outil de développement multi-plates-formes car Visual Age pour Java est écrit partie en Java et partie en code natif Windows.
- Un ORB Corba.

Installation du produit

VisualAge pour Java requiert Windows 98 ou Windows NT/2000. Le produit ne s'installe pas sur Windows 95. La procédure se passe relativement bien sur Windows 98. Vous risquez de rencontrer des problèmes « d'espace d'environnement insuffisant » auxquels il faut remédier en ajoutant la ligne:

```
SHELL=C:\COMMAND.COM /E:32000 /P
```

dans votre fichier config.sys conformément aux recommandations de la documentation électronique.

La documentation

La documentation papier vient sous la forme de 4 livrets. Les 3 premiers livrets sont exclusivement consacrés aux informations relatives à la licence d'utilisation du produit, tout cela étant écrit dans toutes les langues du monde et d'ailleurs. Il est vraiment dommage de constater que 75 % de la documentation papier est finalement consacrée à informer le client de ce qu'il ne peut pas faire plutôt qu'à l'aider à prendre le produit en mains. Lisez tout de même attentivement ces conditions de licence. Vous y apprendrez, par exemple, que vous n'avez pas le droit d'utiliser DB2 en condition de production. D'une manière générale, aucun des outils fournis avec le produit n'est utilisable dans des conditions de production, à moins que vous n'ayez acquis parallèlement les licences d'utilisation correspondantes.
Le quatrième livret est trop succinct à notre goût. Il permet toutefois d'appréhender l'EDI, ce qui dans le contexte n'est pas si mal.
Visual Age vient également avec une abondante documentation électronique comportant de nombreux didacticiels. Nous avons rencontré un problème relatif à cette documentation pour une installation de VisualAge en local. Dans ce cas VisualAge s'obstine à vouloir accéder à la documentation *via* un serveur HTTP comme dans le cas d'une installation distribuée. Convenons que ce problème est peu grave car VisualAge est avant tout un outil de développement destiné à être utilisé dans un environnement distribué.

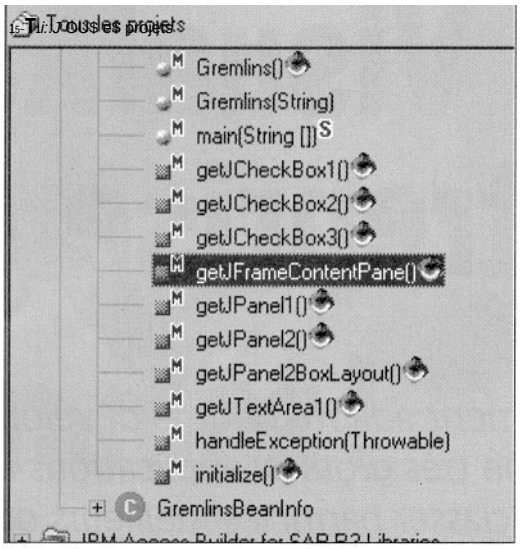
Prise en mains

Tout d'abord, VisualAge surprend par son *look* et son organisation. On ne tarde toutefois pas à comprendre qu'on dispose là d'un EDI d'une très grande qualité, voire somptueux.

La fenêtre principale de l'EDI est organisée en deux couches. La première est nommée espace de travail. La deuxième couche visualise, le cas échéant, le code source partiel de l'élément sélectionné dans la première couche. VisualAge n'affiche jamais le code source complet d'une classe dans sa fenêtre principale. Au début, ceci désarçonne un peu, mais très vite on se rend compte que cette organisation est géniale, car elle clarifie le code Java qui en a bien souvent besoin. Lorsque vous cliquez sur le nom d'une classe dans l'espace de travail, l'éditeur de source ne montre que les membres de la classe (voir la figure 1). .

Ceci donne une vue d'ensemble de la classe, impossible dans d'autres conditions. Si vous développez l'arborescence relative à la classe dans l'espace de travail, vous obtenez une liste des membres et des méthodes de la classe. Chaque élément de la liste est décoré avec une ou plusieurs petites icônes

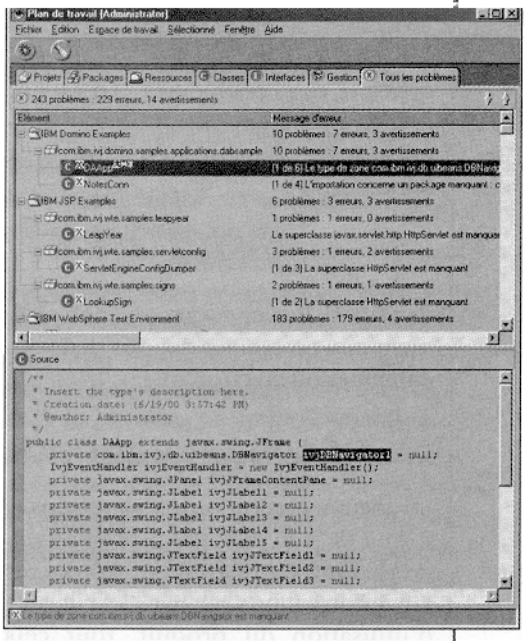
> *Figure 1 : la fenêtre prIncipale de VisualAge avec sa visualisation de code partiel.*



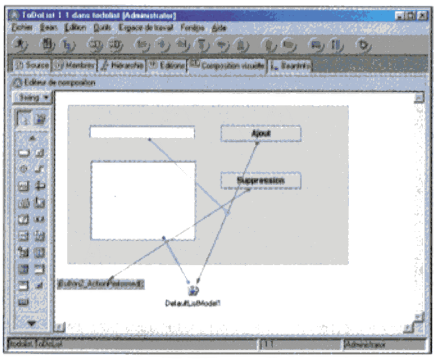
> *Figure 2 : VisualAge décore les noms de méthodes de petites icônes sympathiques.*

indiquant par exemple l'attribut public ou private de l'objet, ou encore, dans le cas d'une méthode, indiquant si celle-ci a été écrite par le programmeur ou automatiquement générée par l'environnement (voir la figure 2). Si vous cliquez sur un nom de méthode, le code de celle-ci, et uniquement lui, apparaît dans la seconde couche.

L'espace de travail peut fournir 7 vues différentes au moyen d'onglets. En plus de l'onglet Projet dont nous venons de parler, il y en a pour visualiser les *packages*, ressources, classes



> *Figure 3 : VisualAge récapitule tous les problèmes d'un espace de travail.*



> *Figure 4.. une applet en cours de création et les connexions entre Beans.*

et interfaces indépendamment. Les plus innovants sont les onglets Tous les problèmes et Gestion.

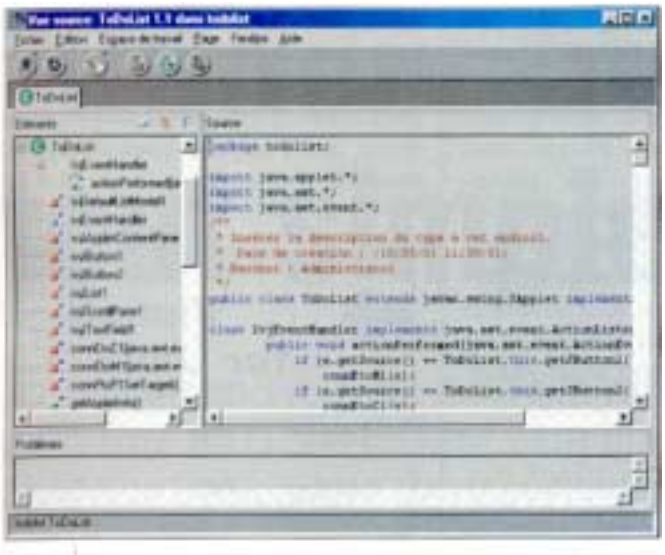
Le premier donne une vue d'ensemble de tout ce qu'il y a à faire pour mettre un projet en ordre (voir la figure 3).

C'est une sorte de gigantesque pense-bête intelligent qui renvoie automatiquement aux extraits de code source concernés. Le développeur a toute liberté de remédier aux problèmes quand il le souhaite. Ceux-ci ne sont jamais bloquants.

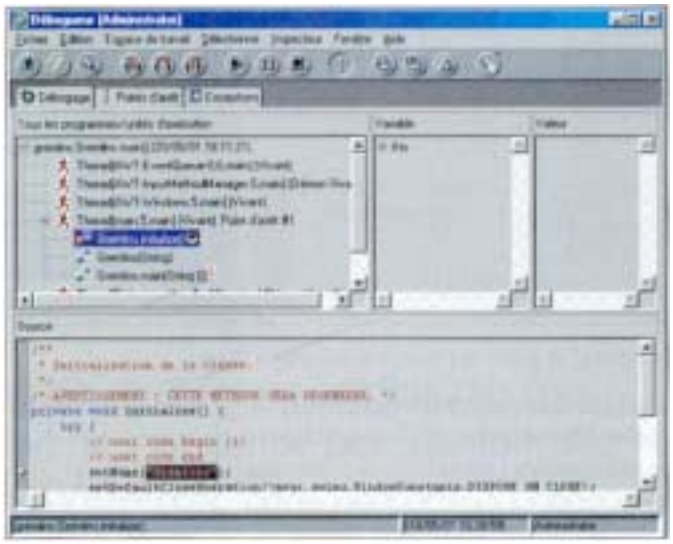
Le deuxième onglet gère l'environnement distribué, permettant de visualiser qui développe quoi et qui est propriétaire de tel ou tel *package*. Signalons à cette occasion que la prise en charge du travail en équipe et du contrôle de version sont vraiment impeccables et sans équivalent dans le monde des EDI Java.

Le cycle de développement

Lorsqu'une classe est créée, VisualAge ouvre automatiquement l'éditeur de composition visuel, même si vous souhaitez créer une classe qui n'a rien de visuel. La philosophie de VisualAge est que tout est *Bean* et qu'un *Bean* peut avoir des connexions avec d'autres, et l'éditeur de composition visuel représente, entre autres, ces connexions, ce qui explique que la création de toute classe passe par lui. La figure 4 montre une *applet* en cours de création avec quelques connexions



> *Figure 5 : l'éditeur de code source.*



> *Figure 6: le débogueur plusieurs de VisualAge permet de déboguer plusieurs processus simultanément.*

inter-beans, dont un *bean* non-visuel : DefaultListModel1.

Nous avons dit que VisualAge ne montre que des extraits de code dans sa fenêtre principale. Il n'en dispose pas moins d'un éditeur de code source très agréable (voir la figure 5).

Les assistants d'achèvement de code et de formatage sont irréprochables. Dans le domaine de l'édition de code, l'EDI apporte un plus très appréciable: à tout moment il est possible de « versionner » le code en cours d'écriture. Ceci permet avec un seul clic de souris de conserver des photographies correspondant à des phases significatives du développement. Il est toujours possible de naviguer entre les différentes versions et de reprendre une version ancienne. De même, il est toujours aisé de naviguer entre les outils de développement dont l'interopérabilité est excellente. Le code écrit et débogué (voir la figure 6), il n'y a plus qu'à déployer.

un outil parfait ?

VisualAge est certes d'une très grande qualité, mais n'est pas exempt de défauts dont certains sont assez agaçants. Quelques exemples: l'outil de composition visuel n'est pas complet. Ainsi, il n'est pas possible de sélectionner le type de bord d'un composant, d'une zone de texte, par exemple, dans l'éditeur de propriétés. Dans le même ordre d'idées, il n'est pas possible de sélectionner le *look and feel* d'une application. Toutes ces tâches doivent être effectuées par un codage manuel ce qui est dommage et nuit à la qualité d'un outil qui se veut dispenser au maximum le dévelop- peur d'écrire du code.

VisualAge vient avec une Jre 1.2.2 de derrière les fagots d'IBM. Nous avons eu droit à un plantage spectaculaire de cette Jre lors de l'exécution depuis l'environnement de l'exemple SwingSet de Sun également fourni avec le produit. Cette Jre sert à l'exécution de tous les composants Java de l'environnement. Or, elle ne dispose manifestement pas de compilateur JIT ce qui rend parfois le maniement d'outils tout-Java un peu énervant en raison du manque de performance. Le plus embêtant est qu'il n'est apparemment pas possible de changer cette Jre, ce qui implique que VisualAge est prévu pour développer avec laJDK 1.2.2 et rien d'autre. Compte-tenu de la perpétuelle mouvance des Jdk et du fait que tout le monde ne peut ou ne veut pas utiliser Java 2, cette restriction fait figure de mouche dans le lait.

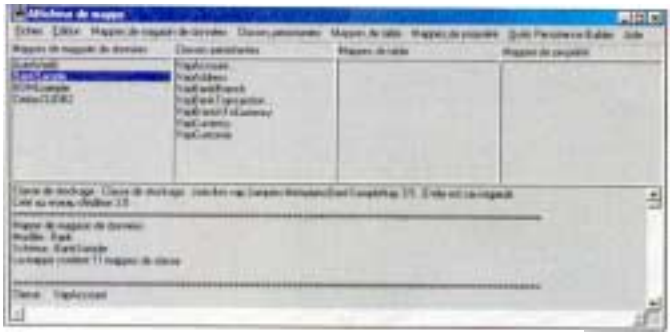
Pas de loi pour les JAR(s)

La plus grosse faiblesse de VisualAge se situe, selon nous, dans le déploiement. VisualAge ne semble pas capable de générer un manifeste pour les archives .jar. Il n'est donc pas possible non plus de spécifier la classe principale d'une archive. D'ailleurs, la notion de classe principale semble complètement étrangère à tout l'EDI. Il n'y a aucune possibilité de déployer à distance, ce qui est d'autant plus paradoxal pour un outil de développement spécialement conçu pour être utilisé en environnement distribué. Il n'est pas possible non plus de déployer un *Entreprise Java Bean* directement sur un serveur d'applications alors que c'est une tâche triviale avec JBuilder d'Inprise, et que cela est également possible avec VisualCafé (WebGain).

D'ailleurs, VisualAge n'est pas fourni avec un serveur d'applications complet. Il est possible de tester EJB et *servlets* sur une émulation de serveur d'applications (WebSphere ou Domino), mais vous ne disposez pas de l'outil de production. Nous pensons qu'il est dommage que WebSphere ne soit pas fourni en version complète, même pour une utilisation limitée au développement. Mais peut-être pense-t-on chez IBM (et sans avoir complètement tort d'ailleurs ;) que Windows n'est pas une plate-forme de production.

Des outils puissants

En plus des outils classiques d'une suite de développement Java, VisualAge dispose de quelques utilitaires particulièrement sympathiques. Ci-



> *Figure 7 : la fenêtre de mappage de persistance Builder*

tons par exemple, Persistence Builder qui *mappe* des objets ainsi que la relation entre ces objets vers des informations stockées dans des bases de données relationnelles (voir la figure 7).
Ou encore, C++ Access Builder qui génère des *wrappers* Java autour de bibliothèques écrites en C++. Ceci permet de réutiliser du code C++ sans devoir s'occuper de programmer la Jni. C++ Access Builder s'avère être un outil précieux pour l'incorporation aisée de moteur de calcul C++ dans les applications Java. C'est encore un moyen de migrer en douceur de C++ vers Java. Signalons enfin les outils spécifiques à IBM. Il s'agit, par exemple, de boîtes à outils pour compiler en code natif; le code Java devant tourner sur plates-formes AS/400 ou OS/390. Il est bien dommage qu'IBM ne fournisse pas un compilateur natif pour plates-formes Windows.

Conclusion

VisualAge est un des tout meilleurs environnements de développement Java sous Windows, ceci malgré ses défauts et malgré le fait qu'il ne soit pas multi-plates-formes ni livré sans serveur d'applications ni ORB Corba. Son EDI est probablement le plus adapté de tous ceux du marché pour l'écriture de grosses et très grosses applications en Java. Pour le développement en équipe et pour le développement d'applications de type e-commerce, VisualAge est un outil d'une très grande valeur. Si vous travaillez dans un contexte « IBM » : plate-formes AS/400 ou OS/390, WebSphere, Domino, etc... VisualAge est le choix qui s'impose tout naturellement. ■